Andrew Adamatzky, Ramon Alonso-Sanz,
Anna Lawniczak, Genaro Juarez Martinez,
Kenichi Morita, Thomas Worsch
Editors

AUTOMATA-2008

Theory and Applications of Cellular Automata

Luniver Press

AUTOMATA-2008
Theory and Applications of Cellular Automata

Andrew Adamatzky, Ramon Alonso-Sanz, Anna Lawniczak,
Genaro Juarez Martinez, Kenichi Morita, Thomas Worsch
Editors

AUTOMATA-2008

Luniver Press
2008

Published by Luniver Press
Frome BA11 3EY United Kingdom

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

AUTOMATA-2008

Cover image: Configuration of two-dimensional cellular automaton Life rule B3/S12345678. Image courtesy of Dr. Genaro Juarez Martinez.

ISBN-10: 1-905986-16-5
ISBN-13: 978-1-905986-16-3

While every attempt is made to ensure that the information in this publication is correct, no liability can be accepted by the authors or publishers for loss, damage or injury caused by any errors in, or omission from, the information given.

Editorial

164. Problem; Ulam
Let a finite number of points, including 0 and 1, be given on the interval $(0,1)$*; a number* $\epsilon > 0$*, and a transformation of this finite set into itself;* T*, with the following property: for every point* p*,* $|p, T(p)| > \epsilon$*. Let us call a "permissible step" passing from the point* p *to* $T(p)$ *or one of the two neighbors (points nearest from the left and from the right side) of the point* $T(p)$*.*
Question: Does there exist a universal constant k *such that there exists a point* p_0 *from which, in a number of allowed steps* $E(\frac{k}{\epsilon})$ *one can reach a point* q *which is distant from* p_0 *by at least* $\frac{1}{3}$*?*

Stanislaw Ulam[1]

The book offers a unique collection of papers presented at the Automata-2008 workshop held in Bristol, June 12-14, 2008. The event was supported by the Engineering and Physical Sciences Research Council (EPSRC), the UK Government's leading funding agency for research and training in engineering and the physical sciences.

Automata 2008 is the 14th workshop in a series of AUTOMATA workshops established in 1995 by members of the Working Group 1.5 (Cellular Automata and Machines) subordinated to Technical Committee 1 (Foundations of Computer Science) of the International Federation for Information Processing (IFIP). The main goal of AUTOMATA workshops is to maintain a permanent, international and multidisciplinary forum for the collaboration of researchers in the fields of Cellular Automata (CA) and Discrete Complex Systems (DCS). Previous workshops took place in Toronto, Canada (2007); Hiroshima, Japan (2006); Gdansk, Poland (2005); Karlsruhe, Germany (2004); Leuwen, Belgium (2003); Prague, Czech Republic (2002); Giens, France (2001); Osaka, Japan (2000); Lyon, France (1999); Santiago de Chile (1998); Gargnano, Italy (1997); Giessen, Germany (1996); Dagstuhl, Germany (1995).

Automata-2008 is the international workshop on cellular automata, an interdisciplinary field, whose general goal might be summarised as the quest for theoretical constructs, computational solutions and practical implementations of novel and powerful models of discrete world. This workshop brought together work that focuses on advanced theoretical constructions, experimental prototypes and implementations of cellular-automaton models, computing devices and paradigms.

[1] English typescript of "The Scottish Book" from the Archive of the Library of Mathematics Faculty of Wrocław University. In: Stefan Banach Portal `http://banach.univ.gda.pl/e-scottish-book.html`

The book presents results of cutting edge research in cellular automata framework of digital physics and modelling of spatially extended non-linear systems; massive-parallel computing, language acceptance, and computability; reversibility of computation, graph-theoretic analysis and logic; chaos and undecidability; evolution, learning and cryptography.

The book will enable researchers, academics and students to get a sense of novel results, concepts and paradigms of cellular automaton theory, delivered by world-leading experts, attract attention of researchers from natural sciences to cost-efficient techniques of cellular-automaton modelling, and beacon industrialists in appreciating high-potential of cellular-automaton computing architectures.

> *Suppose one has an infinite regular system of lattice points in E^n, each capable of existing in various states $S_1, \cdots, S_k$. Each lattice point has a well defined system of m neighbors, and it is assumed that the state of each point at time $t+1$ is uniquely determined by the states of all its neighbors at time t. Assuming that at time t only a finite set of points are active, one wants to know how the activation will spread.*
>
> Stanislaw Ulam[2]

Andrew Adamatzky, Ramon Alonso-Sanz, Anna Lawniczak,
Genaro Juares Martinez, Kenichi Morita, Thomas Worsch
The Conference Organizers
May 2008
Bristol (UK), Madrid (Spain), Guelph (Canada),
Mexico DC (Mexico), Hiroshima (Japan), Karlsruhe (Germany)

[2] Ulam S. M. A Collection of Mathematical Problems (New York: Interscience, 1960), p. 30

Table of Contents

AUTOMATA-2008

Investigations of Game of Life cellular automata rules on Penrose Tilings: lifetime and ash statistics

Nick Owens[1] and Susan Stepney[2]

[1] Department of Electronics, University of York, UK
[2] Department of Computer Science, University of York, UK

Abstract. Conway's Game of Life rules can be applied to Cellular Automata (CAs) running on aperiodic grids, namely Penrose tilings. Here we investigate the result of running such CAs from random initial conditions. This requires development of a Penrose tiling algorithm suitable for CA experiments, in particular, a tiling that can be lazily expanded as CA activity reaches an edge. We describe such an algorithm, our experimental setup, and demonstrate that the Penorse kite and dart tiling has significantly different statistical behaviour from the Penrose rhomb tiling.

1 Introduction

John Horton Conway's *Game of Life* [3][8] is a simple two-dimensional, two state cellular automaton (CA), remarkable for its complex behaviour [3][16]. That behaviour is known to be very sensitive to a change in the CA *rules*. Here we continue our investigations [11] into its sensitivity to changes in the *lattice*, by the use of an aperiodic Penrose tiling lattice [9][14].

Section 2 reviews Penrose tilings, and section 3 describes algorithms to generate them, including one that permits 'lazy extension' of the tiling. Section 4 generalises the concepts of neighbourhood and totalistic CA rules to aperiodic lattices. Section 5 describes the experimental setup for running the Game of Life rules on aperiodic lattices; section 6 reports the statistics of lifetimes, ash densities, and growth of the region of activity.

2 Penrose tilings

2.1 Kites and darts, and rhombs

Grünbaum & Shephard [10, chapter 10] provide a good introduction to aperiodic tilings, including Penrose tilings. The two variants of Penrose tiling we consider here are 'kites and darts', and 'rhombs'.

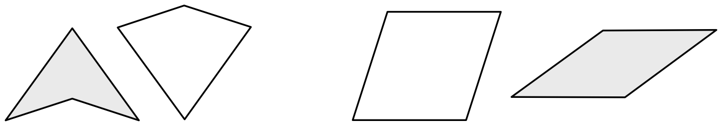


Fig. 1. Penrose tiles (a) the dart (grey) and kite (white) tiles: the long and short sides are in the ratio $\phi : 1$, where the golden ratio $\phi = (1 + \sqrt{5})/2 = 2\cos(\pi/5)$; (b) the thick (white) and thin (grey) rhomb tiles

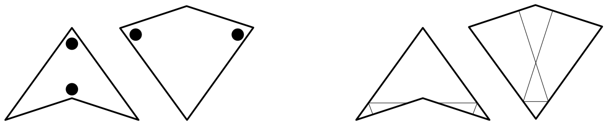


Fig. 2. Kite and dart matching rules (a) vertex markings (b) Ammann bars

The kite and dart tile pair are shown in Fig. 1a; a large patch of kite and dart tiling is shown in Fig. 19. The thick and thin rhomb tile pair are shown in Fig. 1b; a large patch of rhomb tiling is shown in Fig. 21. [21] shows that the number of thick to thin rhombs in a Penrose tiling is in the ratio $\phi : 1$.

2.2 Matching rules and Ammann bars

To avoid a kite and dart being joined to form a rhombus (Fig. 7), and hence a periodic tiling, there are additional 'matching rules': as well as edges of the same length being put together, certain vertices (given by the dots in Fig. 2a) must also be matched [9][10]. Another matching scheme uses Ammann bars (Fig. 2b), which must be joined in straight lines across tiles [10]. (Completed Ammann bars highlight the underlying structure, and can be used to construct tilings; see section 3.5.)

To avoid rhomb tiles being used to form a periodic tiling, there are additional 'matching rules': as well as edges of the same length being put together, the edge orientations (given by the arrows and dots in Fig. 3a) must also be matched [5]. Another matching scheme uses Ammann bars (Fig. 3b), which must be joined in straight lines across tiles.

2.3 Valid vertex configurations

There are many ways to put the tiles together, even with the restriction of the matching rules. However, in a true Penrose tiling (one that can be extended to infinity), not all of these configurations can exist.

There are only seven valid ways to surround any vertex in a kite and dart tiling [9] (Fig. 4).

There are only eight valid vertices in a rhomb tiling [5] (Fig. 5). The names of these vertices come from the names of the corresponding kite and dart vertices

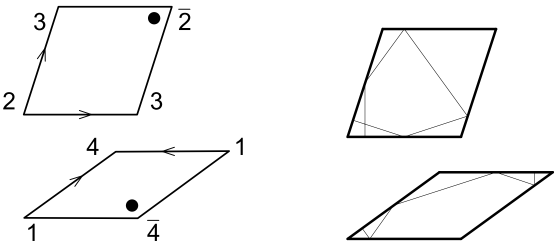


Fig. 3. Rhomb matching rules (a) vertex marking and edge orientations plus vertex angle numbering, where interior angles are $\pi/5$ times the vertex angle number (note that vertices labelled 2, and labelled 4, come in two kinds, due to the matching rules: these are distinguished by overbars) (b) Ammann bars

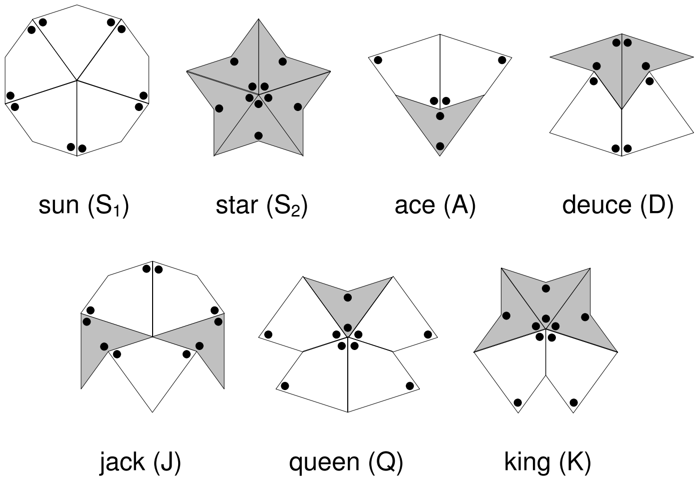


Fig. 4. The seven valid vertex configurations of a kite and dart tiling [9]

from which they can be derived [5]. Each vertex can be associated with a list of vertex angle numbers (after [17, fig.6.8], augmented here with overbars, Fig. 3a), corresponding to the vertex angles of the tiles forming the central vertex. These are useful for determining how to complete forced vertices (see section 3.3). Note that there are two distinct occurrences of the 3,3 vertex configurations (in the J and D); see Fig. 6.

If a patch of tiling exhibits any other vertex configuration, it is not a true Penrose tiling: it will not be possible to extend it to infinity. We use these valid vertex configurations to analyse valid neighbourhood configurations later.

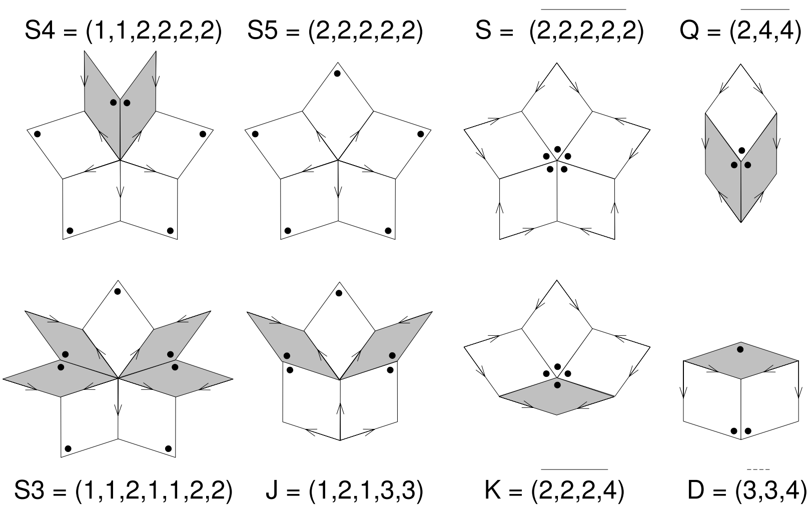


Fig. 5. The eight valid vertex configurations of a rhomb tiling [5]

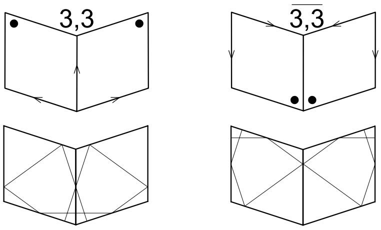


Fig. 6. Disambiguating the 3,3 vertices: the two distinct ways a 3,3 vertex can appear in a valid rhomb vertex configuration (in the J and D, see Fig. 5). This is a *context dependent* marking [21].

3 Penrose tiler construction

3.1 Requirements

There are several different algorithms for constructing valid Penrose tilings of arbitrary size. Here our aim is to experiment with CA rules on a Penrose tiling. We need to decide what to do as activity approaches the edge of the current grid. One common solution to this is to implement a 'lazy' grid, which expands as necessary. This provides requirements for a Penrose tiling algorithm suitable for investigating CAs:

1. The lazy tiler shall tile an arbitrarily sized rectangular region with a valid Penrose tiling (supporting both kites and darts, and rhombs)

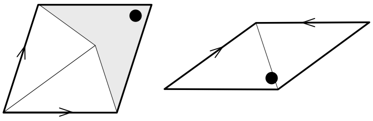


Fig. 7. Relationship between (marked) rhomb tiles and kites and darts: a thick rhomb comprises a dart and two half-kites; a thin rhomb comprises two half-kites

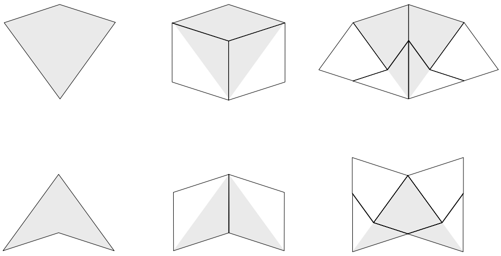


Fig. 8. One iteration of deflating a dart tile and a kite tile, via intermediate rhomb tiles (after [10, Figure 10.3.19]). The ratio of the original and final kite and dart tile sizes is the golden ratio ϕ.

2. Given any point on the plane outside the tiled region, the tiler shall appropriately expand the tiled region to a larger rectangle that includes this point
3. During such expansion, the tiler shall not extend tiling unnecessarily (neither in directions away from the expansion point, nor beyond it)
4. The tiler shall determine the neighbourhood of each tile, for use as a CA lattice
5. The tiler shall determine when CA activity reaches the edges of the currently defined lattice, and trigger suitable grid expansion

Our pentagrid lazy tiling algorithm meets these requirements. We describe it here in some detail, because previous descriptions of pentagrid tiling algorithms are somewhat obscure (and not tuned for CA experiments), and the lazy extension algorithm has not been described before.

3.2 Deflation

The relationship of the rhomb tiles to the kite and dart tiles is shown in Fig. 7. The deflation tiling method involves recursively breaking tiles into sub-tiles (Fig. 8). This is one of the best known method of creating a valid Penrose tiling, and derives from methods used in [14].

Ramachandrarao *et al* [15] describe a related decomposition process that produces a valid rhomb tiling starting from a single thick rhomb, where the decomposition processes are “akin to normal crystallographic operations”.

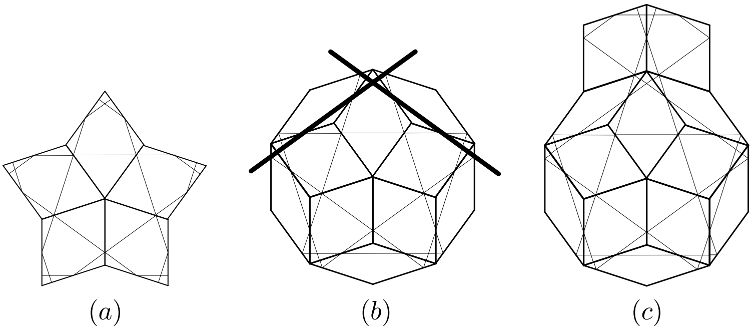


Fig. 9. Example of Onoda *et al* [13] rhomb tiling algorithm. (a) Start from an S vertex initial seed cluster; each concave vertex on the tiling edge is $(\bar{3}, \bar{3})$, so is a forced D vertex. Apply *R1* to completed each with a thin rhomb. (b) The result is a regular decagon, with no further forced vertices. The underlying Amman bars intersect at 108° (bold lines). *R2* requires a thick tile consistent with the vertex rules to be added "*to either side*" (that is, to one side or the other) of the corresponding tiling vertex, which here has (partial) vertex number list (1,2,1) (c) Both S3 and J have a partial vertex list of (1,2,1), but extension with a *thick* tile implies extension with a 2, 3 or $\bar{3}$ vertex. The only way this can be done consistently is to complete it as a J vertex, (1,2,1,3,3).

The deflation approach was taken in [11]; as noted there it is not a suitable process for a lazy tiler, since the generation n grid does not clearly appear as a subpart of the larger generation $n + 1$ grid. So although it is possible to create arbitrarily large tilings (requirement 1), it is not possible to do so 'lazily' (requirement 2); this size must be set at the start.

3.3 Onoda's rhomb tiling algorithm

Onoda *et al* [13] describe an algorithm for generating rhomb tilings. Start from any valid 'seed cluster' tiling (for example, a single tile, or one of the eight valid vertex tilings in Fig. 5). "*R1: If one or more vertices are forced, choose a forced vertex and add a forced tile to it.*" A forced vertex is one that can be completed in only one way to give a valid vertex. (Without loss of generality, we can add the entire set of tiles needed to complete the forced vertex.) "*R2: If there are no forced vertices, add a thick tile (consistent with the vertex rules) to either side of any* 108° *corner.*" The 108° corner referred to in this rule is not (necessarily) a 3-vertex of a thick rhomb: it is rather the underlying corner defined by the edges of the completed forced patch of tiling "*when viewed macroscopically*". These macroscopic edges are traced out by the underlying Amman bars in the matching rules (Fig. 9). Again, without loss of generality, we can add the entire set of tiles needed to complete the chosen vertex.

This process results in a steadily growing patch of tiling (requirements 1 and 2). However, it is not suitable for a lazy tiler, since the direction that the tiling grows is not under programmatic control: it has to grow in the direction of forced vertices first, then 108° corners (so does not satisfy requirement 3).

3.4 Ammann bars

The underlying Ammann bars of a Penrose tiling form five grids of lines, each grid rotated by a multiple of $2\pi/5$ (Fig. 9). Given such a grid, the underlying tiling can be reconstructed.

A lazy tiler based on Ammann bars could satisfy requirement 3, as new grid lines could be laid down only in the required direction. However, these grid lines are not evenly spaced: they form a Fibonacci sequence [21]. The lazy tiling extension (R2), which involves laying down new grid lines, would be possible, but slightly tricky. Since there is a similar approach, but based on a regularly-spaced pentagrid (see next section), the Ammann bar approach is not considered further here. ([21] shows that if one starts with a rhomb tiling reconstructed from Ammann bars, and deflates it, then the resulting rhomb tiling is one that can be reconstructed from a regular pentagrid.)

3.5 Pentagrid

A multigrid comprises sets of evenly spaced parallel lines all at appropriate rotations of each other. de Bruijn [6] shows that a multigrid is the dual of a tiling. (The dual tiling of a grid associates a tile with every intersection of the grid, and a vertex of the tile with every open area of the grid.) A special case of the multigrid is the *pentagrid*, and its dual is a Penrose rhomb tiling [5].

The description of how to extract the dual rhomb tiling from a pentagrid given in this section is adapted and simplified from [5] [21]. de Bruijn [5] provides the original pentagrid definitions and theorems. His formulation is in terms of complex numbers (rather than vectors), and has no diagrams, which can make it hard to understand the approach in algorithmic terms. Socolar & Steinhardt [21] recast the approach in terms of vectors, but also generalise to multigrids and three dimensions, which again can obscure the components of a tiling algorithm. Austin [1] provides a good overview of the technique.

A *grid* (Fig. 10a) is a sequence of regularly spaced parallel lines, labelled with the integers. A unit vector $\mathbf{e}$ normal to the lines defined the direction of the grid. A number γ defines the distance of the line labelled zero from the origin. So the grid line labelled N is defined by

$$\mathbf{x}.\mathbf{e} = N + \gamma \tag{1}$$

where N is the line label.

A point $\mathbf{x}'$ not on a grid line is assigned an index value equal to the higher label of the adjacent grid lines:

$$N_{x'} = \lceil \mathbf{x}'.\mathbf{e} - \gamma \rceil \tag{2}$$

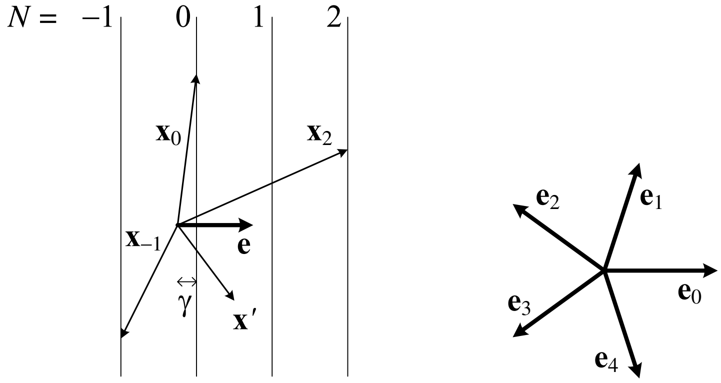


Fig. 10. (a) A grid defined by unit vector $\mathbf{e}$ and offset γ (b) Five unit vectors, with angles $2k\pi/5 = k \times 72°$, defining a pentagrid. Note that $\mathbf{e}_0$ and $\mathbf{e}_1$ form two edges of a thick rhomb, and $\mathbf{e}_0$ and $\mathbf{e}_2$ form two edges of a thin rhomb. Note also that $|\mathbf{e}_0 + \mathbf{e}_1 + \mathbf{e}_2| = \phi$.

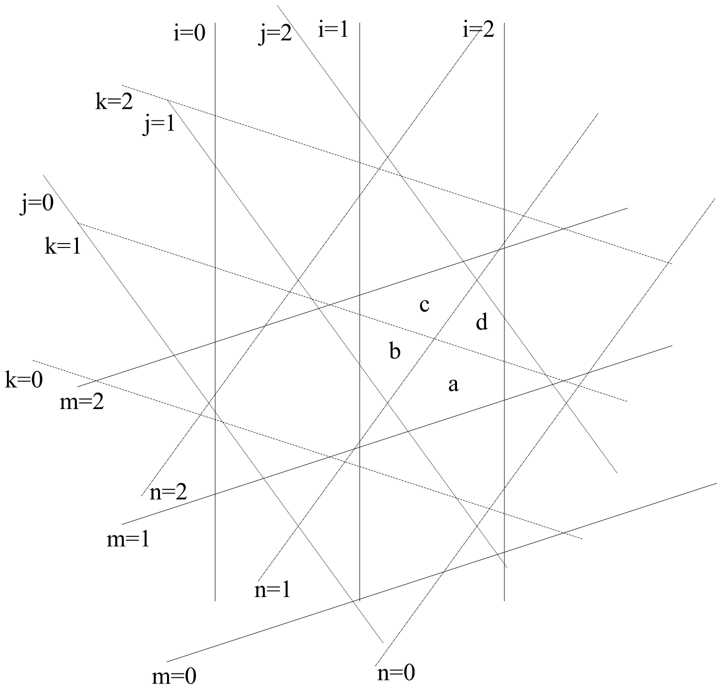


Fig. 11. A regular pentagrid. Each polygon formed by the grid lines is represented by the 5-tuple of grid labels (i, j, k, m, n): $a = (2, 2, 1, 2, 1)$; $b = (2, 2, 1, 2, 2)$; $c = (2, 2, 2, 2, 2)$; $d = (2, 2, 1, 2, 2)$. Note that adjacent polygons differ in only one label value (corresponding to the grid containing the grid line forming their common edge).

A *pentagrid* is a set of five grids, defined by five equally spaced direction vectors (Fig. 10b), and five offsets $\gamma_0, \ldots, \gamma_4$. de Bruijn [5] defines a *regular pentagrid* to be one where at most two grid lines cross at any one point (Fig. 11), achieved by suitable choice of offsets. (A sufficient condition for regularity is where all the offsets $-1 < \gamma_i < 1$, all are different, $\gamma_0 = 0$, and neither $\gamma_1 + \gamma_4$ nor $\gamma_2 + \gamma_3$ is an integer.)

Then, under the condition that

$$\sum_{i=0}^{4} \gamma_i = 0 \tag{3}$$

de Bruijn [5] proves that the pentagrid is the dual of a Penrose rhombus tiling: a tile is defined at every intersection of grid lines, and a tile vertex at every open polygonal region surrounded by grid lines; the condition that no more than two lines intersect ensures a unique tiling. ([21] extends some of these definitions and results to grids with irregular spacings of grid lines, including Ammann bar grids, and to 3D grids.)

Consider a pentagrid with each of its open regions labelled with a 5-tuple of integers $(k_0, k_1, k_2, k_3, k_4)$, calculated according to equation 2 (see Fig. 11). Each such region corresponds to the vertex of a rhomb, with coordinates [5, eqn 5.2] [21, eqn 4(−1)]

$$\mathbf{v} = \sum_{i=0}^{4} k_i \mathbf{e}_i \tag{4}$$

de Bruijn [5, eqn 5.4] defines

$$I = \sum_{i=0}^{4} k_i \bmod 5 \tag{5}$$

to be the *index* of the rhomb vertex, and shows that it is never 0.

The regularity constraint ensures that precisely four regions surround any intersection of pentagrid lines. These four regions differ in only two of their five grid labels (corresponding to grids containing the two intersecting grid lines they surround), and the values of these labels differ by at most one (Fig. 11). From equation 4 and Fig. 10b, we can therefore see that any such four adjacent regions correspond to the vertices of a thick rhomb (if the grid lines intersect at an angle of $2\pi/5$) or a thin rhomb (if the grid lines intersect at an angle of $4\pi/5$).

The fact that the rhombs so defined form a valid Penrose tiling is the basis of the pentagrid tiling algorithm.

3.6 Pentagrid lazy tiler algorithm

The pentagrid approach satisfies our requirements for a lazy tiler.

Requirement 1: Given a rectangular region to be tiled, T, we can lay a pentagrid over it, and extend the lines of the pentagrid to calculate all intersections that lie within T.

Requirement 2: Given a point $p \notin T$, we can extend the region to be tiled to T', the smallest rectangular region that includes both T and p, and then extend the pentagrid to calculate the new intersections that lie within the extended region $T' - T$.

This also satisfies requirement 3, that the tiling is not unnecessarily extended during this process.

Requirement 4: The algorithm calculates the vertices of rhombs before it calculates the rhombs themselves. These vertices are also vertices of neighbouring rhombs, so the neighbourhood can be built up as the tiles are calculated.

Requirement 5 can be satisfied as follows. Consider some further region definitions. The *neighbourhood* of a tile t, $N(t)$, is the set of all tiles close to t in some way (different kinds of Penrose tiling neighbourhoods suitable for defining CAs are given in section 4.1). Let C be the 'complete neighbourhood' region: all tiles t in T whose neighbourhood is 'complete' (that is, also in T) for some neighbourhood definition $N(t)$.

$$C = \{\ t \in T \mid N(t) \subseteq T\ \} \tag{6}$$

Clearly $C \subseteq T$, and unless the neighbourhood is trivial, $C \subset T$. Let A be the region where we allow cellular automaton activity to exist. We require

$$A \subseteq C \subset T \tag{7}$$

The extension of A is dictated by development of cellular automaton activity. This can be triggered when there is activity, a change in state, of a tile $t \in A$ where $N(t) \nsubseteq A$, so there are members of the neighbourhood of t outside A (although they will be in T if the requirement of equation 7 holds before this activity step). We extend A to include all the neighbourhood of t. In order to ensure that $A \subseteq C$ still holds, this may require extension of C, and hence of T.

It would be possible to calculate this extension exactly; however it is far cheaper in terms of computational complexity to extend the tiling such that the distance between the boundary of T and A is always greater than a distance d that ensures that the condition $A \subseteq C$ holds. (This does slightly violate requirement 2, that the extension not go beyond p, but not to any significant degree.)

For a generalised Moore neighbourhood (suitable for Game of Life CA rules) the value of d is determined by examining the valid vertex configurations in Figs. 4 and 5. For kites and darts, the largest is the queen (Q) vertex (Fig. 4), with a maximum diameter of twice the width of a dart, or $4\sin(2\pi/5)$ times the length of a short edge. For rhombs, the largest is the S3 vertex (Fig. 5), with a maximum diameter of twice the long diagonal of the thin rhomb, or $4\sin(2\pi/5)$ times the length of an edge.

The original treatment [6] uses the pentagrid approach to produce a rhomb tiling; here we outline how it is used in an algorithm for a lazily extending tiling, of rhombs, and of kites and darts.

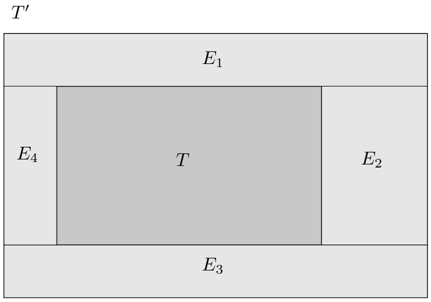


Fig. 12. Initial tiling T extends to tiling T' with extension regions E_1, E_2, E_3, E_4

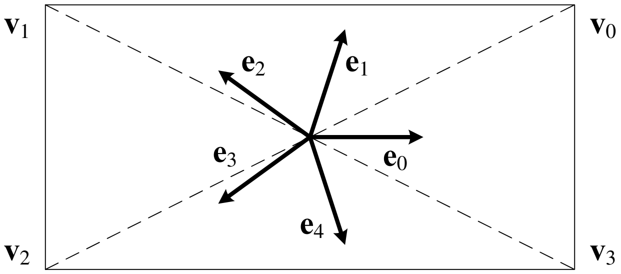


Fig. 13. An arbitrary rectangle R to be tiled, with corners $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$, and diagonals shown with dashed lines. For each $\mathbf{e}_i$, the diagonal with direction closer to that of $\mathbf{e}_i$ gives the two opposite corners between which grid lines should be calculated. So $\mathbf{v}_0$ and $\mathbf{v}_2$ are used to define the extent of grid lines for $\mathbf{e}_1$ and $\mathbf{e}_3$; $\mathbf{v}_1$ and $\mathbf{v}_3$ for $\mathbf{e}_2$ and $\mathbf{e}_4$ either pairs of opposite corners may be used for $\mathbf{e}_0$.

Choosing the offsets. First, choose suitable offsets γ_i that ensure a regular pentagrid and that obey equation 3. A sufficient such choice is $(0, \gamma, -\gamma, 2\gamma, -2\gamma)$, governed by the single parameter γ, where $0 < \gamma < 0.5$.

In all the results reported here, we used the offsets $(0.2, -0.2, -0.1, 0.4, -0.3)$. This violates the $\gamma_0 = 0$ condition, but does create regular pentagrid.

Tiling a region. We consider only the tiling of a rectangular region. The problem of tiling an initial region and then extending the tiling can be reduced to tiling arbitrary rectangles. For example, Fig. 12 shows the extension of tiling T to tiling T' with four rectangular extension regions E_1, E_2, E_3, E_4.

To tile an arbitrary rectangle R we must ensure that all grid intersections that lie within the rectangle are calculated. For each of the five grids, every one of its lines that intersects the rectangle must be considered. We do this by observing that the maximum extent of the grid that needs to be considered is defined by the rectangle's diagonal that is closer to the direction of the grid, $\mathbf{e}$ (Fig. 13). So, for example, grid 1 has the set of grid values N_1 given by (using

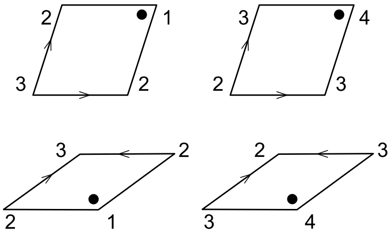


Fig. 14. Marking a rhombus depending on its vertex indices. These are the only combinations of vertex indices that are generated from a pentagrid. (Note: these vertex index numbers are unrelated to the 'vertex angle numbers' shown in Fig. 3a.)

equation 2)

$$\lceil \mathbf{v}_2.\mathbf{e}_1 - \gamma_1 \rceil < N_1 < \lceil \mathbf{v}_0.\mathbf{e}_1 - \gamma_1 \rceil \tag{8}$$

All pairs drawn from $N_0 \times N_1 \times N_2 \times N_3 \times N_4$ calculated in this way contain all the grid intersections in the rectangle R (plus more outside).

Converting grid intersections to tile vertices. To calculate the vertices of a tile we must discover the 5-tuples defining the four open regions around an intersection. The 5-tuples corresponding to each intersection index are calculated using equation 2 (to get the other three indices of the intersection 5-tuple) and Fig. 11 (to get the four vertex 5-tuples). The coordinates of the corresponding vertex are given by equation 4.

Iterating over intersections naively would result in each vertex being calculated 4 times: to ensure the each vertex is calculated only once, we store the vertex information in a hashtable indexed by its 5-tuple. At each vertex we also maintain a list of all tiles that share the vertex: this aids subsequent neighbourhood calculations.

Conversion from rhomb to kite and dart tiling. The pentagrid approach yields a rhomb tiling. This can subsequently be converted to a kite and dart tiling using the mapping shown in Fig 7.

First, the rhombs need to be "marked", in a way that depends on the vertex index (equation 5). de Bruijn [5] shows that the only combinations of vertex indices formed by a pentagrid are $(1, 2, 2, 3)$ or $(2, 3, 3, 4)$; the marked vertex is the one with vertex index 1 or 4 (Fig. 14).

Each thick rhomb is broken into a dart and two half-kites, each thin rhomb is broken into two half-kites (Fig. 7), taking the rhomb marking into account. The half-kites are then suitably joined to produce a complete kite and dart tiling.

Precision issues. We have the condition that only two grid lines cross at any intersection point. de Bruijn [7] gives the condition for a singular pentagrid

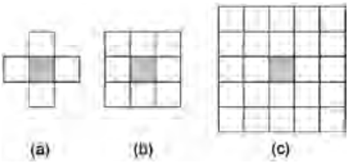


Fig. 15. (a) von Neumann neighbourhood (b) Moore neighbourhood, or box neighbourhood of radius $r = 1$ (c) box neighbourhood of radius $r = 2$

(where more than two lines intersect at some point) to be for grid $i \in \{0, 1, 2, 3, 4\}$ and an integer k if

$$(k - \gamma_i)\phi + \gamma_{i-1} + \gamma_{i+1} \in \mathbb{Z} \tag{9}$$

Choosing grid $1 = 0$, $\gamma_0 = 0$, this gives

$$k\phi + \gamma_1 + \gamma_4 \in \mathbb{Z} \tag{10}$$

Assuming non-extreme choices of γ_1 and γ_4 (that is, choosing $\gamma_1 + \gamma_4 = O(1)$), and given $\phi = O(1)$, this condition could occur in a computation by rounding error, if there were an error of one unit in the last place of k.

Our implementation uses Java doubles, which are implemented using double-precision 64-bit IEEE 754 floating point, which have a 53-bit mantissa, corresponding to approximately 16 decimal digits. So the algorithm does not lose accuracy (no grid singularity occurs) below $N = O(10^{16})$ grid lines, or $O(10^{32})$ tiles. If larger tilings than this are required, higher precision arithmetic should be used. (This is only an order of magnitude argument, but since our tiling investigations are significantly below this limit, with $O(10^5)$ tiles, we can be confident that there is no loss of accuracy affecting our results.)

4 Cellular automata on aperiodic lattices

Classic cellular automata are defined on regular lattices. The update rule depends on the state of each cell's full neighbourhood (the surrounding cells, and the updating cell itself)[3], and the structure of that neighbourhood is invariant: all places in the lattice look the same, and the update rule can be applied uniformly across the lattice. Typical neighbourhoods for 2D cellular automata are shown in Fig. 15. These neighbourhoods can be formally defined in terms of metrics on the lattice. However, we define them (later) in an equivalent manner that permits easy generalisation to aperiodic lattices.

[3] The standard definition of CA neighbourhood includes both the surrounding cells *and* the updating cell. Throughout this paper we use slightly different terminology (because we are nearly always referring to outer totalistic CA rules, see later): by *neighbourhood* we mean only the surrounding cells. If we want to include the updating cell, we refer to the *full neighbourhood*.

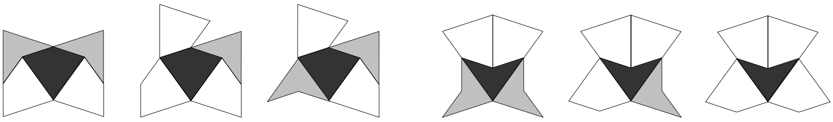


Fig. 16. The generalised von Neumann neighbourhoods of a kite and dart Penrose tiling.

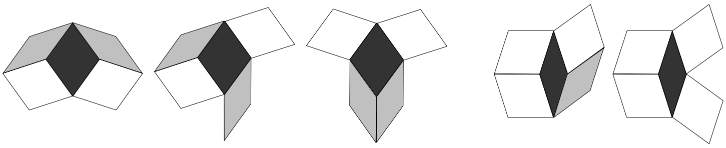


Fig. 17. The generalised von Neumann neighbourhoods of a rhomb Penrose tiling.

In general, the update rule depends on the particular state of each separate neighbour. For *totalistic* CA rules, however, the next state of a cell depends only on the number of full neighbourhood cells in certain states. For *outer totalistic* CA rules, the next state of a cell depends only on its current state, and the number of neighbourhood cells in certain states.

For example, in Conway's *Game of Life* outer totalistic CA, the neighbourhood of each cell comprises the 8 nearest cells of the regular Moore neighbourhood (Fig. 15b). Each cell has two states, 'dead' and 'alive'. If a cell is alive at time t, then it stays alive iff it has 2 or 3 live neighbours (otherwise it dies of 'loneliness' or 'overcrowding'). If a cell is dead at time t, then it becomes alive (is 'born') iff it has exactly 3 live neighbours.

For aperiodic lattices such as a Penrose tiling, the detailed structure of the neighbourhood varies at different locations in the lattice. Totalistic and outer totalistic rules can be given an interpretation in these aperiodic tiling neighbourhoods.

4.1 Generalised von Neumann neighbourhood

We define the generalised von Neumann neighbourhood of a cell to be all the cells with which it shares an edge (or, equivalently, two distinct vertices). Hence the size of the neighbourhood equals the number of edges of the central cell. Figures 16 and 17 show the range of distinct generalised von Neumann neighbourhoods which form valid vertices (rotations and mirror images of these neighbourhoods are not considered to be distinct).

de Bruijn [7] identifies the same rhomb neighbourhoods (but considers mirror images separately), and shows that a valid Penrose rhomb tiling can be constructed by considering just these neighbourhoods, without the need to use the rhomb matching rules of Fig. 3.

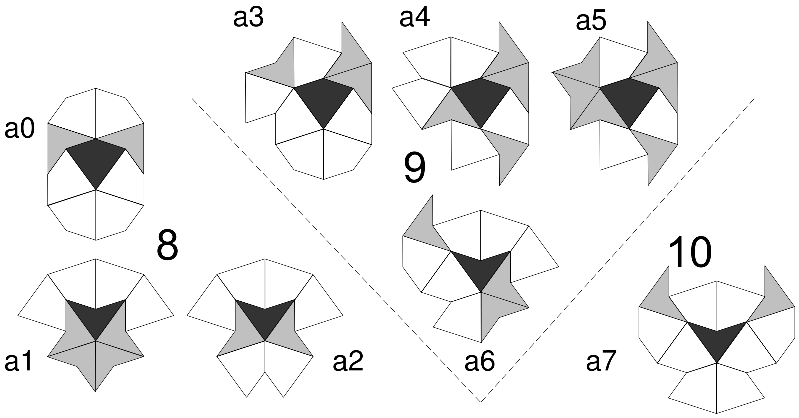


Fig. 18. The generalised Moore neighbourhoods on a kite and dart Penrose tiling, with neighbourhood sizes.

In the rectangular lattice (Fig. 15a), none of the four von Neumann neighbourhood cells themselves share an edge. So if A is a neighbour of B, and B is a neighbour of C, then A is *not* a neighbour of C: neighbouring von Neumann neighbourhoods do not overlap (recall that we do not treat the central site as a member of the neighbourhood for the purposes of this paper). In the Penrose lattice, this is no longer the case: cells in a generalised von Neumann neighbourhood can share an edge, so neighbouring generalised von Neumann neighbourhoods can overlap. This may affect the communication paths through the Penrose CA.

4.2 Generalised Moore neighbourhood

We define the generalised Moore neighbourhood of a cell to be all the cells with which it share a vertex.

Not only do cells have irregular shaped neighbourhoods, with the generalised Moore neighbourhood not all cells have the same number of neighbours. The range of neighbourhood sizes and configurations is limited.

Fig. 18 shows the eight distinct generalised Moore neighbourhoods in a kite and dart tiling: there are no other valid ways to surround a kite or a dart (this can be established by exhaustive consideration of the valid vertex configurations shown in Fig. 4). So there is one neighbourhood configuration of size 8 around a kite, and two around a dart; three of size 9 around a kite, and one around a dart; and one of size 10, around a dart. ([11] incorrectly states that kite and dart tilings have neighbourhoods of size 8 and 9 only.) Figure reffigure:moorekitenbrhood show an area of kite and dart tilings with colouring to highlight the size of cells' neighbourhoods.

Similarly, Fig. 20 shows the 11 distinct generalised Moore neighbourhoods in a rhomb tiling. There is a larger range of distinct neighbourhood configurations

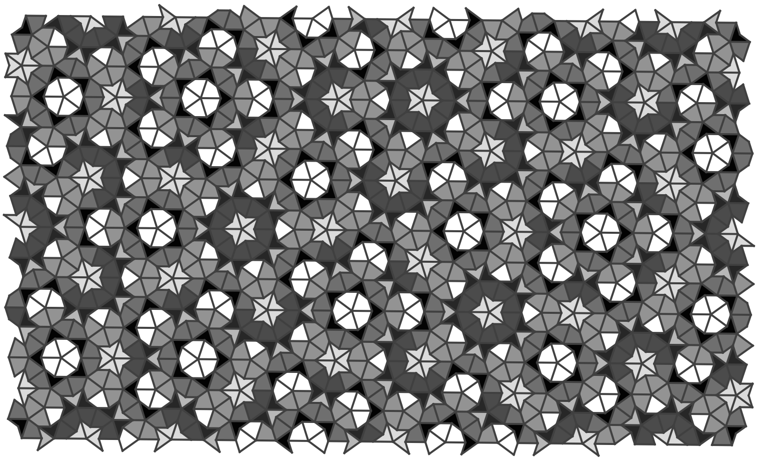


Fig. 19. A kite and dart tiling shaded by neighbourhood type. The neighbourhood shading is uniformly distributed between white and black such that a_0 is white and a_7 black.

for rhomb tilings. Figure 19 show an area of rhomb tilings with colouring to highlight the size of cells' neighbourhoods.

As can be seen from Figs. 19 and 21, not all sizes of neighbourhoods appear with the same frequency. Figure 22 shows the distribution of neighbourhood sizes in a kite and dart tiling and in a rhomb tiling.

4.3 Generalised box neighbourhood

The Moore neighbourhood is a special case of a box neighbourhood, with radius $r = 1$. Let $N(c)$ be the generalised Moore neighbourhood of cell c. We define recursively a generalised box neighbourhood of radius $r > 1$. Let $N(c, r)$ be the box neighbourhood of cell c, of radius r. Then define

$$N(c, 1) = N(c) \tag{11}$$

$$N(c, r) = \bigcup_{n \in N(c)} N(n, r - 1) \tag{12}$$

See Figs. 23 and 24.

The frequency distribution of neighbourhood sizes for $r = 2$ neighbourhoods is shown in Fig. 25 for kites and darts, and for rhombs. Again, the rhomb tilings have larger neighbourhoods, and a larger spread of neighbourhood sizes.

4.4 Penrose Life rules

Using our definition of the generalised Moore neighbourhood, the definition of the Game of Life as given in section 4 can be used unchanged on a Penrose

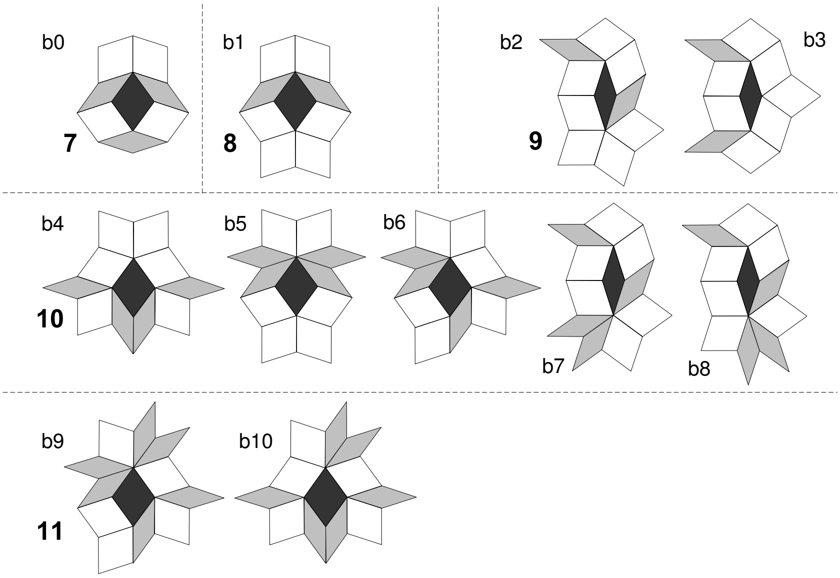


Fig. 20. The generalised Moore neighbourhoods on a rhomb Penrose tiling, with neighbourhood sizes.

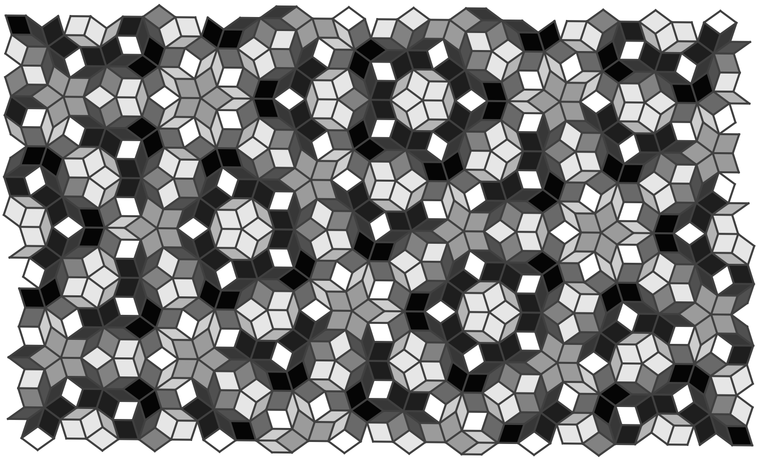


Fig. 21. A rhomb tiling shaded by neighbourhood type. The neighbourhood shading is uniformly distributed between white and black such that b_0 is white and b_{10} black.

		kite/dart				rhomb			
size	type	cells	%			type	cells	%		
7						b0	2831	9.1		
									2831	9.1
8	a0	4994	14.7			b1	4576	14.6		
	a1	4248	12.5							
	a2	1890	5.6							
				11132	32.7				4576	14.6
9	a3	6116	18.0			b2	2134	6.8		
	a4	6125	18.0			b3	2842	9.1		
	a5	3762	11.1							
	a6	3774	11.1							
				19777	58.2				4976	15.9
10	a7	3083	9.1			b4	2370	7.6		
						b5	1735	5.6		
						b6	2133	6.8		
						b7	3475	11.1		
						b8	3501	11.2		
				3083	9.1				13214	42.3
11						b9	3522	11.3		
						b10	2136	6.8		
									5658	18.1

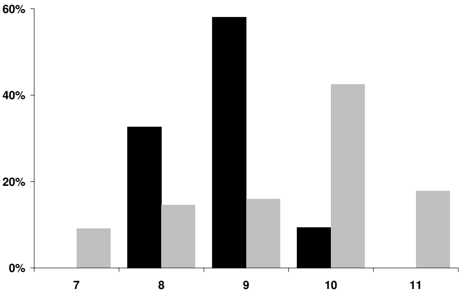


Fig. 22. Generalised Moore neighbourhood statistics, on a 33992 cell kite and dart tiling (black bars, median size = 9), and a 31255 cell rhomb tiling (grey bars, median size = 10)

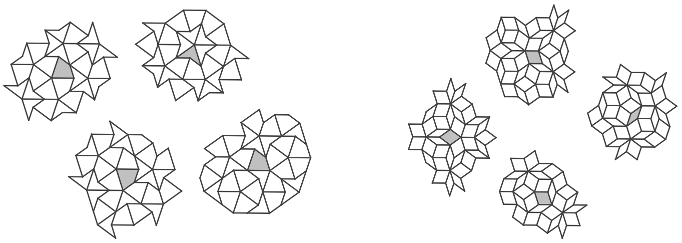


Fig. 23. Examples of box radius $r = 2$ aperiodic neighbourhoods on kite and dart Penrose tilings, and on rhomb Penrose tilings

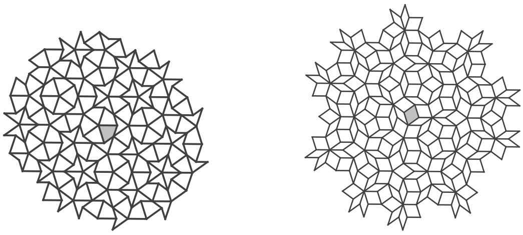


Fig. 24. An example of a box radius $r = 5$ aperiodic neighbourhood on a kite and dart Penrose tiling, and on a rhomb Penrose tiling

size	kite/dart		rhomb	
	cells	%	cells	%
25	4248	12.5		
26	4994	14.7	1597	5.1
27	4204	12.4	829	2.7
28	11330	33.3	1912	6.1
29	2351	6.9	1092	3.5
30	3782	11.1	1977	6.3
31	1441	4.2	1314	4.2
32	1642	4.8	2588	8.3
33			493	1.6
34			4609	14.7
35			1075	3.4
36			4622	14.8
37			3489	11.2
38			828	2.6
39			4830	15.5

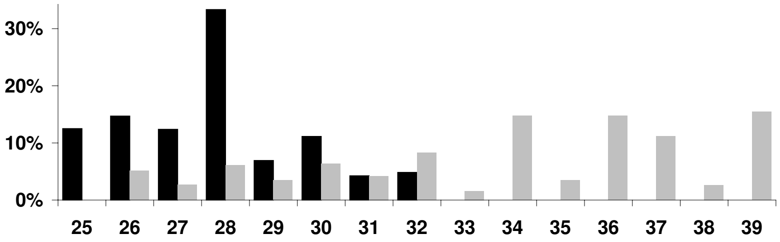


Fig. 25. Neighbourhood statistics $r = 2$ neighbourhood, on a 33992 cell kite and dart tiling (black bars, median size = 28), and a 31255 cell rhomb tiling (grey bars, median size = 34)

lattice. Some early investigations are reported in [11]; further investigations are reported later in this paper.

In our investigations, we use some typical GoL terminology, defined here. (The quoted definitions are from [19].)

soup "*A random initial pattern, often assumed to cover the whole Life universe.*" Here we consider only finite soup extents, but allow subsequent activity outside the initial soup patch.

quiescence Eventual periodic CA activity. Once the CA has entered a quiescent state, its future activity is periodic, and hence predictable.

ash "*The (stable or oscillating) debris left by a random reaction.*" Hence an ash is the quiescent state left by a soup.

5 Experimenting with Life

In [11] we report that the Game of Life has different quantitative behaviour on a regular lattice and on a Penrose kite and dart lattice: on the Penrose lattice the lifetime to quiescence is much shorter, and the ash density is lower. This section investigates if there are similar differences between the behaviour of the rules running on kite and dart and on rhomb lattices.

Null Hypothesis: The Game of Life run on kites and darts has identical statistical behaviour to the Game of Life run on rhombs.

To test this hypothesis, we investigate three statistics: lifetime to quiescence, ash density, and growth of the active area.

5.1 Experimental setup

To test the hypothesis we vary the density D of soups of similar sizes S on rhomb and kite and dart tilings, run the cellular automaton to quiescence, and record the lifetime to quiescence t_q, ash density ρ (measured over the soup box), and soup growth g.

Lifetime t_q: The lifetime, or the time to quiescence, is defined to be the number of timesteps from the soup state ($t = 1$) until the pattern of live cells (measured over the whole timing G) first repeats (at $t = t_q$). Each timestep, the CA's current state is stored, along with the number of live cells. To check for quiescence, the current state is compared to all previous states with the same number of live cells. The period p is the number of timesteps since the state was previously seen: $p = t_q - t_{prev}$.

Ash density ρ: The proportion of live cells in the ash at $t = t_q$, measured over the soup tiles.

Soup growth g: The number of cells in the maximum active area divided by the number of cells in the soup: $g = A/S$. measured over the soup tiles (Fig. 26).

Tiling grid: We use a lazily expanding tiling for both kites and darts, and rhombs. We use an initial tiling of size $G = 23194$ for the kite and dart experiments, and of size $G = 23123$ for the rhomb experiments. It is difficult to produce

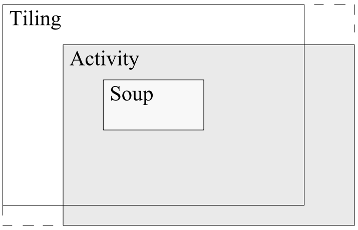


Fig. 26. The initial tiling grid G, the central soup area S, the maximum activity area during the run A, and the possibly extended tiling grid G_q (dashed box) to accomodate the activity.

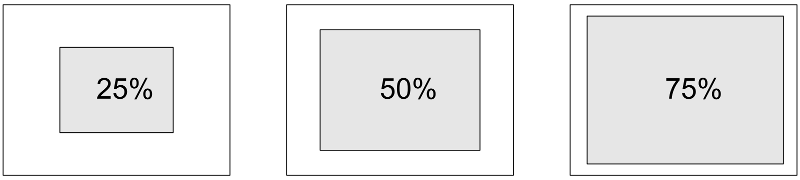


Fig. 27. The three central soup areas, to scale with the initial grid area.

identical sized tilings: these are deemed close enough for fair comparison. These differences in tile numbers are of similar scale to the differences in tile numbers between regular and kite and dart tilings used in [11] (and are about twice the size of the largest grid explored there).

Soup area: Three initial soup areas S, covering the central 25%, 50%, 75% of the area of the tiling. See Fig. 27 and 28.

Soup density: 100 soup densities D, in the range $[0.01, 1.0]$ with increments of 0.01. Each cell in the soup area S is initially alive with probability D; all other cells in G are initially dead. See Fig. 29.

Runs: Each of the 100 soup densities D across the three soup sizes S is run to quiescence 1000 times.

5.2 Statistical analysis

We want to test whether certain distributions are statistically the same or different: the commonly-used tests assume an underlying normal distribution. Are the distributions here (sufficiently) normal?

Figures 31 and 32 show the histograms of lifetime and ash density results over the 1000 runs for one particular soup size and soup density. The lifetime distributions, at least, do not look normal.

We investigate further the distribution of lifetimes and ash densities for these examples. We calculate the median, mean, standard deviation, skew and kurtosis of these distributions (using the MS-Excel functions MEDIAN, AVERAGE, STDEV, SKEW, and KURT respectively), for the lifetimes (Fig. 33) and the ash densities

	25%	50%	75%	G
kite and dart	5842	11670	17527	23194
rhomb	5815	11611	17405	23123

Fig. 28. Number of tiles involved in the experiments, soup sizes S = 25%, 50% and 75%, and full initial grid size G

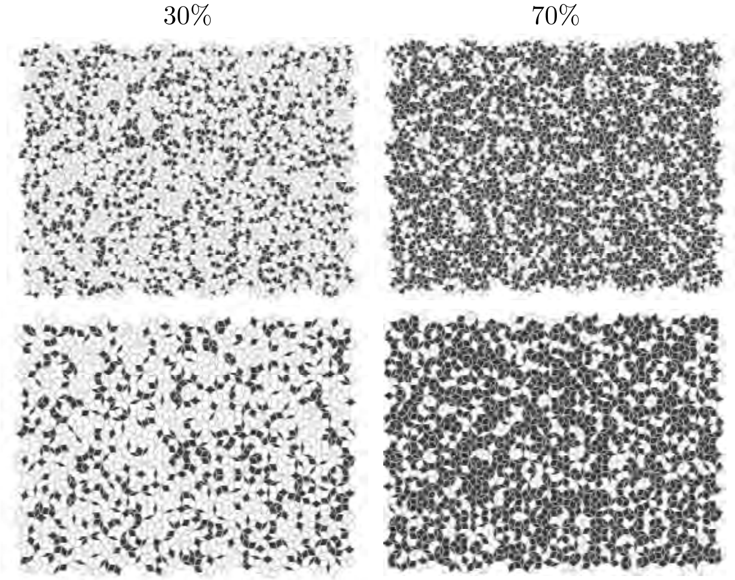


Fig. 29. Typical soups at two densities, for kite and dart (top) and rhomb (bottom) tilings

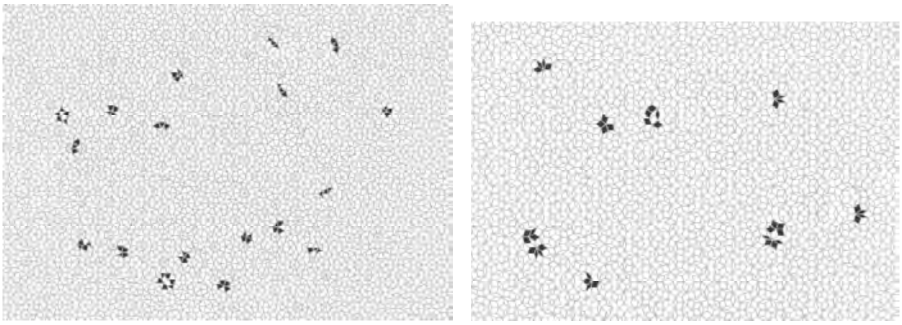


Fig. 30. The ashes resulting from the 30% soups of Fig. 29. Note the extended areas of activity.

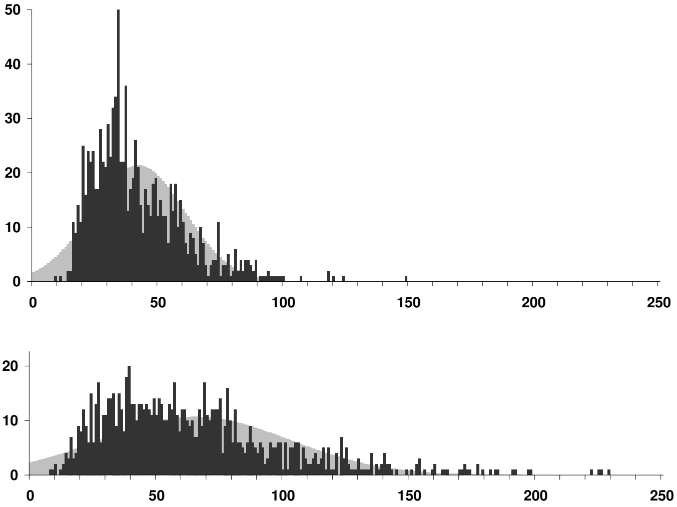


Fig. 31. The distribution of lifetimes to quiescence on the kite and dart tiling (top) and rhomb tiling (bottom), for 1000 runs with soup size $S = 25\%$ and soup density $D = 0.8$; with comparison normal distributions of the same mean and standard deviation.

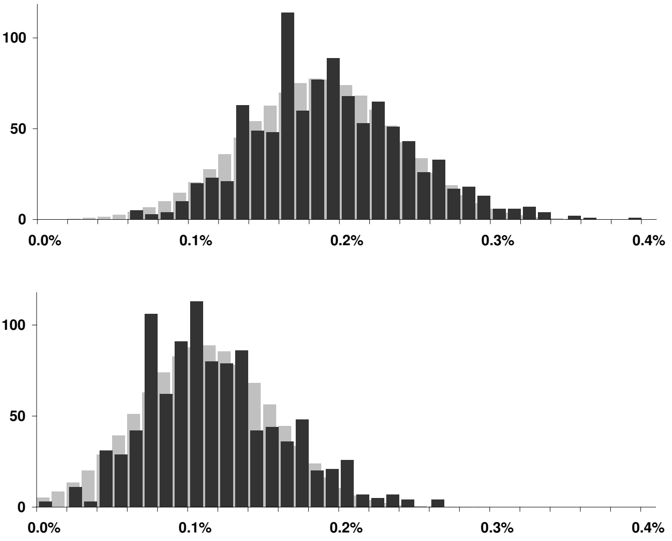


Fig. 32. The distribution of ash densities on the kite and dart tiling (top) and rhomb tiling (bottom), for 1000 runs with soup size $S = 25\%$ and soup density $D = 0.8$; with comparison normal distributions of the same mean and standard deviation.

		$D = 0.4$		$D = 0.8$	
soup		k&d	rhomb	k&d	rhomb
25%	m	96	158	37	57.5
	μ	99.4	163.0	41.9	65.1
	σ	19.6	37.0	18.6	37.3
	s	1.1	0.9	1.2	1.5
	k	2.0	1.2	2.1	3.9
50%	m	108	179	40	60
	μ	111	185.1	44.7	66.6
	σ	19.6	37.2	18.5	33.7
	s	0.7	0.8	1.2	1.2
	k	0.9	0.5	2.0	2.1
75%	m	116	190	44	67
	μ	118.6	198.1	47.1	74.1
	σ	20.0	40.4	17.9	35.9
	s	0.9	1.2	1.3	1.2
	k	1.1	2.8	4.3	4.3

Fig. 33. Statistics for the lifetime distributions (median m, mean μ, standard deviation σ, skew s, kurtosis k) for soup densities $D = 0.4$ and 0.8; soup sizes $S = 25\%$, 50% and 75%

		$D = 0.4$		$D = 0.8$	
soup		k&d	rhomb	k&d	rhomb
25%	μ	0.0044	0.0034	0.0018	0.0011
	σ	0.0008	0.0008	0.0005	0.0004
	s	0.2	0.3	0.4	0.5
	k	0.2	−0.0	0.2	0.1
50%	μ	0.0084	0.0022	0.0063	0.0015
	σ	0.0011	0.0006	0.0010	0.0005
	s	−0.1	0.2	0.1	0.3
	k	0.1	−0.1	−0.0	0.0
75%	μ	0.0123	0.0091	0.0029	0.0023
	σ	0.0027	0.0013	0.0007	0.0006
	s	−0.1	0.2	0.2	0.1
	k	−0.1	−0.1	0.0	−0.1

Fig. 34. Statistics for the ash densities (mean μ, standard deviation σ, skew s, kurtosis k) for soup densities $D = 0.4$ and 0.8; soup sizes $S = 25\%$, 50% and 75%

	kites	rhombs
$\# > 45$	$349\ (= A)$	$650\ (= B)$
$\# \leq 45$	$651\ (= C)$	$350\ (= D)$

Fig. 35. The number of measurements above, and not above, the joint median value of 45, for soup size $S = 25\%$, density $D = 0.8$

(Fig. 34: we do not show medians here, since they are indistinguishable from the means).

For large samples ($N > 150$) drawn from a normal population, the skewness statistic is approximately normally distributed with mean 0 and standard deviation $s_s = \sqrt{6/N}$ [20, §5.13]; for very large samples ($N > 1000$) drawn from a normal population, the kurtosis statistic is approximately normally distributed with mean 0 and standard deviation $s_k = \sqrt{24/N}$ [20, §5.14]. Hence skew values beyond two standard errors of skewness, or kurtosis values beyond two standard errors of kurtosis, indicate that the distribution is not normal at the 95% confidence level.

For $N = 1000$ (just valid for the kurtosis test), $2s_s = 0.5$ and $2s_k = 1.0$. Both these values are lower than those calculated for the lifetimes (Fig. 33), so the lifetime distributions are not normal at the 95% confidence level. Normality of the ash densities has not been ruled out by this test (Fig. 34).

Given this non-normality of the lifetimes, we calculate the non-parametric median and quartile statistics of the runs, for the range of soup densities (Figs. 37 and 39). These results are in qualitative agreement with those in [11]: low lifetimes and ash densities at extreme soup densities; a 'plateau' in the behaviours for soup densities $\sim 0.2 - 0.6$; lifetimes $\sim 100 - 200$; ash densities $\sim 1 - 2\%$. We now, however, have better statistics, and new results for rhomb tilings.

Since the lifetime distributions are not normal, we use the non-parametric *median test*, to test whether the distributions have statistically significantly different medians [18, pp.111–115]. (In practice, our sample sizes are probably large enough that assuming normality and using a t-test is probably valid. However, the certainly valid, if somewhat weaker, non-parametric test is adequate in this case.)

Null Hypothesis T: for soup size $S = 25\%$, density $D = 0.8$, there is no difference between the median lifetimes for kites and darts, and for rhombs.

The calculation involves splitting the measurements into four groups: those above, and not above, the joint median of the measurements. The relevant numbers for our test case are given in Fig. 35). Then we calculate the value of χ^2 from [18, eqn(6.4)]:

$$\chi^2 = \frac{N\,(|AD - BC| - N/2)^2}{(A+B)(C+D)(A+C)(B+D)} = 90.3 \tag{13}$$

The probability of occurrence under Null Hypothesis T for $\chi^2 \geq 90.3$ with one degree of freedom is $p < \frac{1}{2}\,\textsc{chidist}(90.3,1) = 10^{-21}$ for a one-tailed test.

Therefore we can reject Null Hypothesis T, with an *extremely* high degree of statistical confidence.

In fact, the difference in the medians in this test case is statistically significant to an almost ludicrous degree. This extreme level of statistical confidence is due mostly to the large number of samples, $N = 1000$. (Such large samples are much more typical in computer science than, say, medicine, because computer experiments are relatively cheap, and have no ethical considerations.) As Bakan says ([2, ch.1, p.7], as quoted in [12]): "there is really no good reason to expect the null hypothesis to be true in any population". A sufficiently large sample size will always be able to refute a null hypothesis: the smaller the effect, the larger the sample required to detect it. For normally-distributed populations with means and standard deviations similar to those of Fig. 34, sample sizes in the low tens would be sufficient to establish a statistically significant difference of their means at the 99% confidence level.

Because of this, we also perform a test of the *effect size*. We use Cohen's effect size d-test [4, §2.5]. (Strictly speaking, we should not use this statistic, because the distributions are not normal. But if we get a sufficiently large value of the d-statistic, we can still be confident in the importance of the difference.) For samples with different variances but the same sample size, we use

$$d = \frac{m_1 - m_2}{\sqrt{(s_1^2 + s_2^2)/2}} \tag{14}$$

where the m_i are the two sample means, and the s_i are the two sample variances. So d measures the difference in the means compared to the spread of the data.

Cohen's criterion is that $d = 0.2$ indicates a small effect, $d = 0.5$ a medium effect, and $d = 0.8$ a large effect. For soup size $S = 25\%$, density $D = 0.8$, we have $d = 0.8$ indicating a large effect from the change in the tiling.

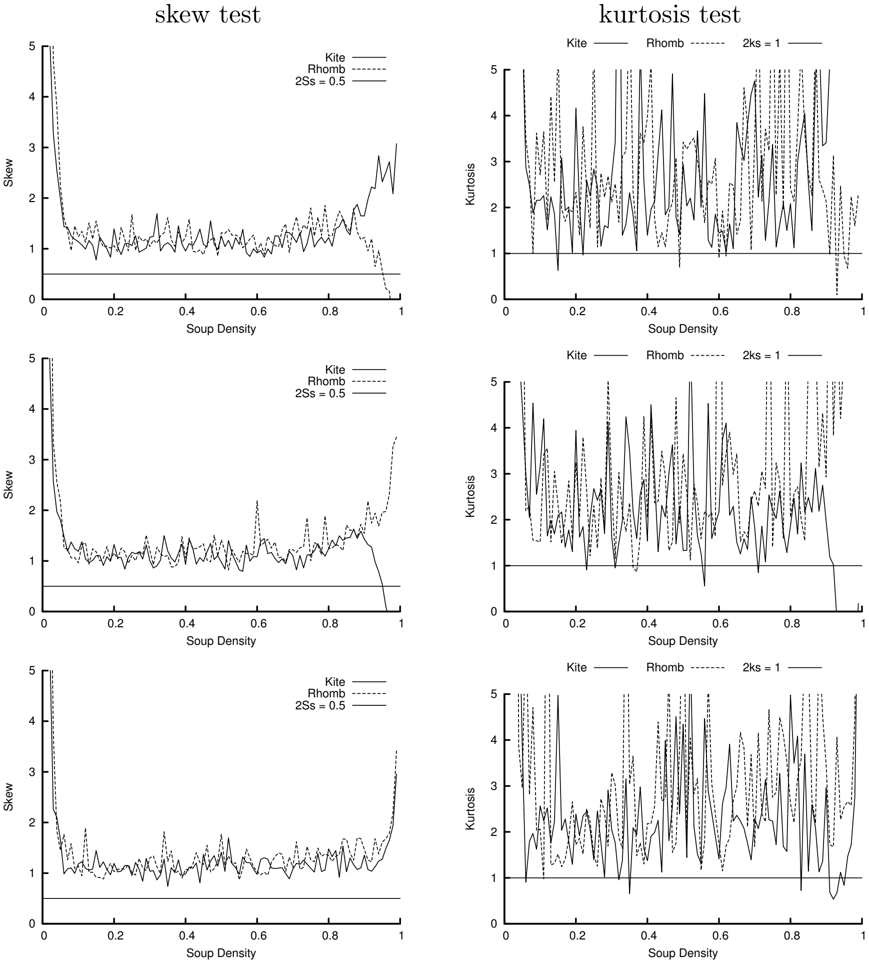
So, for all the results that follow, we do not present the statistical significance: the differences are all extremely significant. We present the skew and kurtosis normality tests, median and quartiles, means, and the effect size, demonstrating that all the statistics chosen exhibit a large effect with the change in the tiling.

6 Lifetime, ash, growth results

6.1 Lifetimes

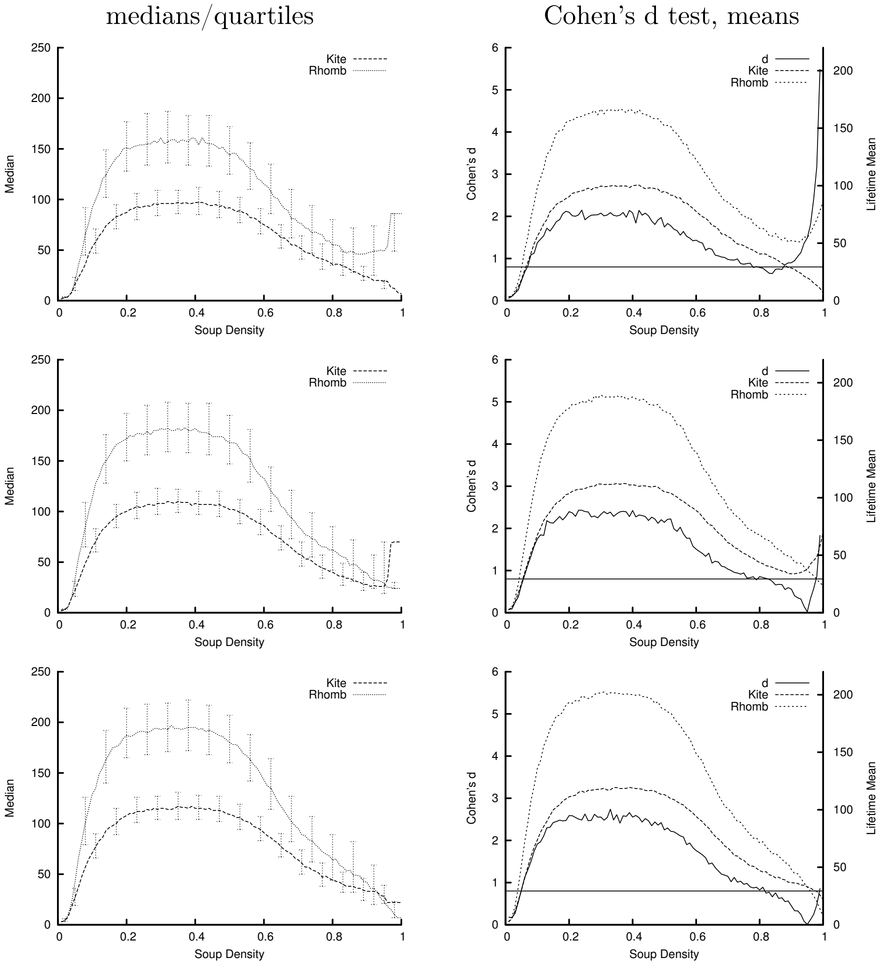
Null Hypothesis 1: The Game of Life run on kites and darts has identical lifetime statistics to the Game of Life run on rhombs.

See Figs. 36, 37. The skew and kurtosis tests show that the distributions are significantly non-normal. The lifetime distributions for the two tilings are different, with a large effect size, refuting Null Hypothesis 1. The Game of Life on the rhomb tiling has significantly longer lifetimes than it does on the kite and dart tiling. From [11], we can say that they both have shorter lifetimes than Life on a regular lattice.


skew test

kurtosis test

Fig. 36. Lifetime to quiescence t_q: normality tests (soup sizes 25% top, 50% middle, 75% bottom)



medians/quartiles

Cohen's d test, means

Fig. 37. Lifetime to quiescence t_q: medians and effect size (soup sizes 25% top, 50% middle, 75% bottom)

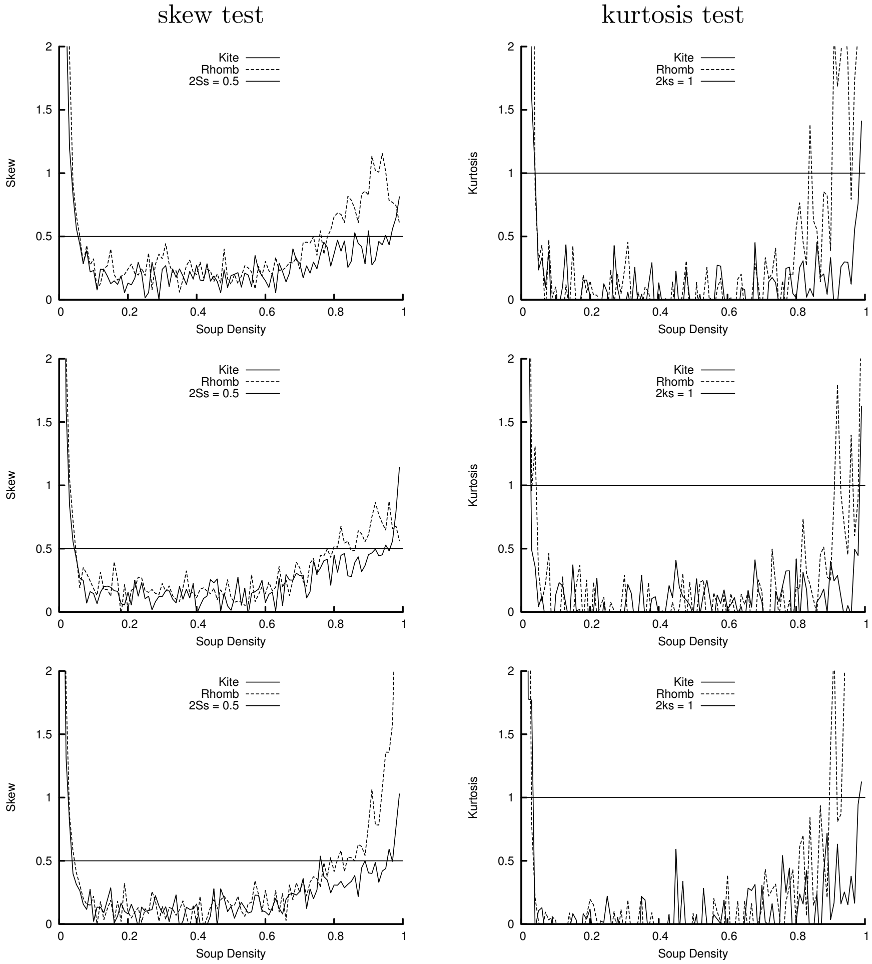


Fig. 38. Ash density ρ: normality tests (soup sizes 25% top, 50% middle, 75% bottom)

6.2 Ash densities

Null Hypothesis 2: The Game of Life run on kites and darts has identical ash density statistics to the Game of Life run on rhombs.

See Figs. 38, 39. The skew and kurtosis tests show that the distributions are consistent with being normal, except for large soup densities. The ash density distributions for the two tilings are different, with a large effect size, refuting Null Hypothesis 2. The Game of Life on the rhomb tiling has significantly lower ash densities than it does on the kite and dart tiling. From [11], we can say that they both have lower ash densities than Life on a regular lattice.

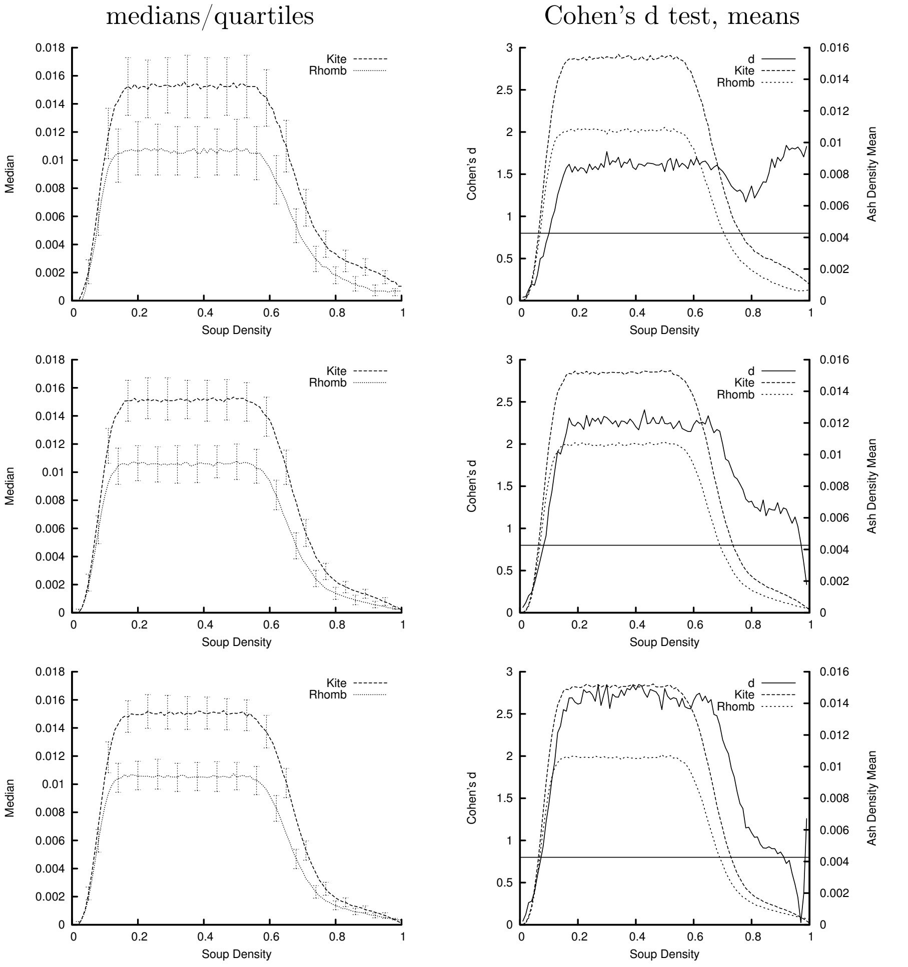


Fig. 39. Ash density ρ: medians and effect size (soup sizes 25% top, 50% middle, 75% bottom)

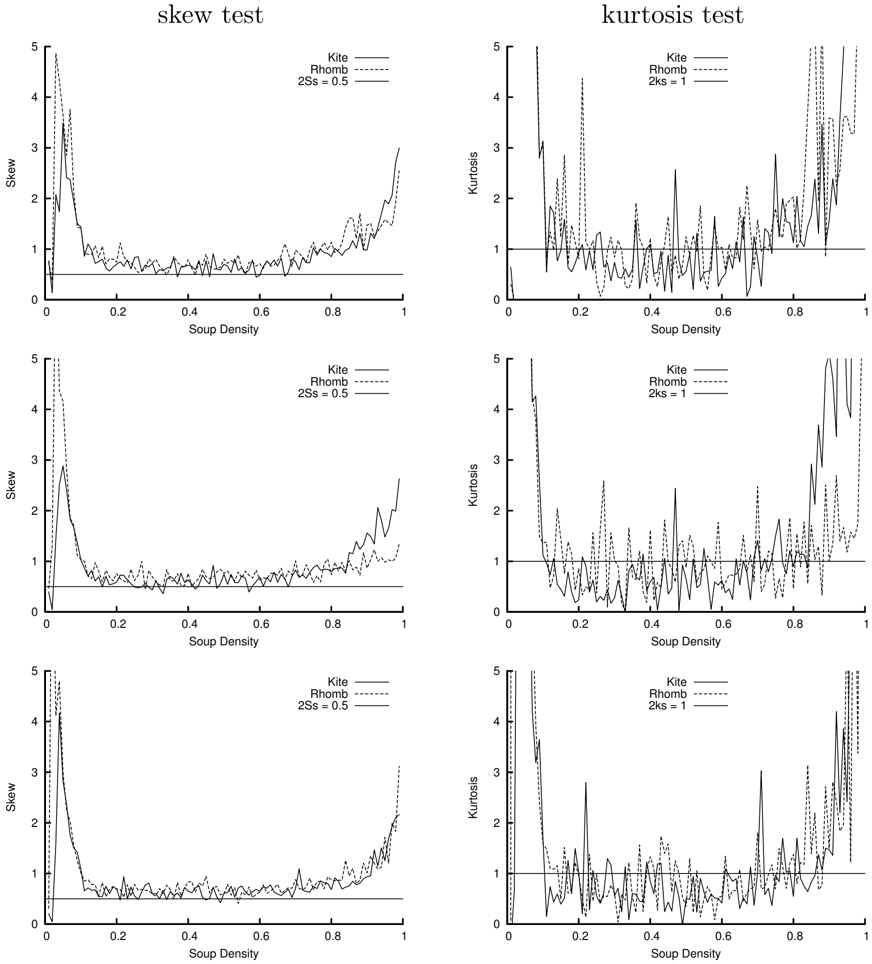
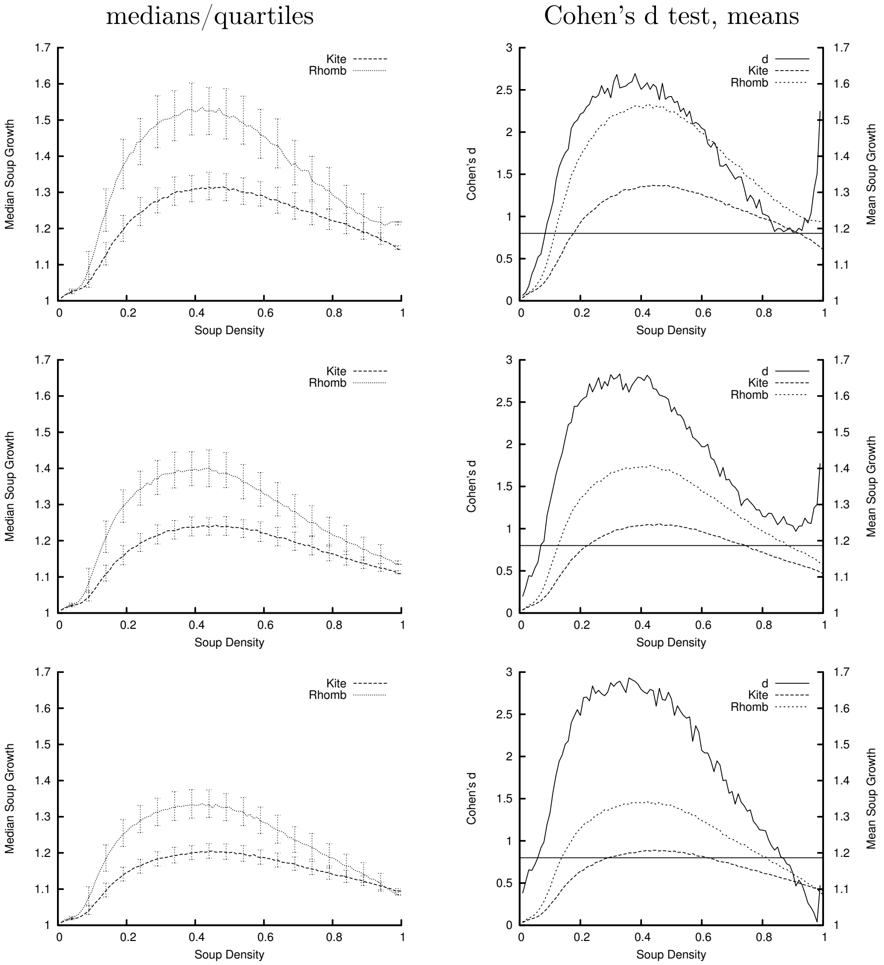


Fig. 40. Soup growth g: normality tests (soup sizes 25% top, 50% middle, 75% bottom)

6.3 Soup growth

Null Hypothesis 3: The Game of Life run on kites and darts has identical growth of soup to the Game of Life run on rhombs.

See Figs. 40, 41 for statistics on the growth of the area of soup. The skew and kurtosis tests show that the distributions are significantly non-normal. The growths of the two tilings are different, with a large effect size, refuting Null Hypothesis 3. The Game of Life on the rhomb tiling has significantly more growth from soup than it does on the kite and dart tiling.


medians/quartiles

Cohen's d test, means

Fig. 41. Soup growth g: median and effect size (soup sizes 25% top, 50% middle, 75% bottom)

7 Conclusions

We have presented a Penrose lazy tiling algorithm, suitable for statistical experiments of CA rules. We have used it to perform experiments with Game of Life rules, and demonstrate that the Game of Life on the rhomb tiling is significantly different from that on the kite and dart tiling: it has longer lifetimes, lower ash densities, and higher soup growth.

Work is underway to investigate and classify the oscillators left in the ash.

References

[1] David Austin. (December 2005). Penrose tilings tied up in ribbons. *AMS Monthly Essays on Mathematical Topics.* `http://www.ams.org/featurecolumn/archive/ribbons.html`.

[2] D. Bakan. (1967). *On Method: Toward a Reconstruction of Psychological Investigation.* Josey-Bass.

[3] Elwyn R. Berlekamp, John Horton Conway, and Richard K. Guy. (1982). *Winning Ways for Your Mathematical Plays Volume 2: games in particular.* Academic Press.

[4] Jacob Cohen. (1988). *Statistical power analysis for the behavioral sciences.* Lawrence Erlbaum Associates, 2nd edition.

[5] N. G. de Bruijn. (1981). Algebraic theory of Penrose non-periodic tilings of the plane I and II. *Indagationes Mathematicae (Proceedings)*, 84:39–66.

[6] N. G. de Bruijn. (1986). Dualization of multigrids. *Journal de physique, Colloque C3*, 47:9–18.

[7] N. G. de Bruijn. (1996). Remarks on Penrose tilings. In R. L. Graham and J. Nesetril, editors, *The Mathematics of P. Erdös, volume 2*, pages 264–283. Springer.

[8] Martin Gardner. (October 1970). Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223(4):120–123.

[9] Martin Gardner. (January 1977). Mathematical games: extraordinary nonperiodic tiling that enriches the theory of tiles. *Scientific American*, 236(1):110–121.

[10] Branko Grünbaum and G. C. Shephard. (1987). *Tilings and Patterns.* W. H. Freeman.

[11] Margaret Hill, Susan Stepney, and Francis Wan. (2005). Penrose Life: ash and oscillators. In Mathieu S. Capcarrere, Alex A. Freitas, Peter J. Bentley, Colin G. Johnson, and Jonathan Timmis, editors, *Advances in Artificial Life: ECAL 2005, Canterbury, UK, September 2005*, volume 3630 of *LNAI*, pages 471–480. Springer.

[12] Marks R. Nester. (1996). An applied statistician's creed. *Applied Statistics*, 45(4):401–410.

[13] George Y. Onoda, Paul J. Steinhardt, David P. DiVincenzo, and Joshua E. S. Socolar. (1988). Growing perfect quasicrystals. *Physical Review Letters*, 60(25):2653–2656.

[14] Roger Penrose. (1978). Pentaplexity. *Eureka*, 39:16–32.

[15] P. Ramachandrarao, G. V. S. Sastry, L. Pandev, and Arvind Sinha. (1991). A novel algorithm for a quasiperiodic plane lattice with fivefold symmetry. *Acta Cryst.*, A47:206–210.

[16] Paul Rendell. (2002). Turing Universality of the Game of Life. In Andrew Adamatzky, editor, *Collision-Based Computing*. Springer.

[17] Marjorie Senechal. (1995). *Quasicrystals and Geometry*. Cambridge University Press.

[18] Sidney Siegel. (1956). *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill.

[19] Steven Silver. (February 2006). Life lexicon, release 25. `http://www.argentum.freeserve.co.uk/lex.htm`.

[20] George W. Snedecor and William G. Cochran. (1980). *Statistical Methods*. Iowa State University Press, 7th edition.

[21] Joshua E. S. Socolar and Paul J. Steinhardt. (1986). Quasicrystals. II. Unit-cell configurations. *Phys. Rev. B*, pages 617–647.

A new universal cellular automaton on the pentagrid

Maurice Margenstern[1] and Yu Song[1]

Laboratoire d'Informatique Théorique et Appliquée, EA 3097,
Université de Metz, I.U.T. de Metz,
Département d'Informatique,
Île du Saulcy,
57045 Metz Cedex, France,
margens@univ-metz.fr

Abstract. In this paper, we significantly improve a result of the first author, see [1]. In the quoted paper, published in 2003, the authors show the existence of a weakly universal cellular automaton on the pentagrid with 22 states. The simulation used in [1] uses a railway circuit which simulates a register machine. In this paper, using the same simulation tool, we lower the number of states for a weakly universal cellular automaton down to 9.

1 Introduction

As indicated in the abstract, this paper significantly improves a previous result.

The result published in [1] was indeed the first universal cellular automaton devised in a hyperbolic space. The simulation was performed on the pentagrid and, as mentioned in the abstract, it uses a simulation of a register machine through a railway circuit. The simulation of [1] requires 22 states.

We refer the reader to [3, 1] for the simulation of a register machine by a railway circuit. In order to help the reader to better understand the paper, here we sketch out the main lines of this simulation. First, we indicate the guidelines of the general simulation, and then we turn to the implementation in the pentagrid.

2 The railway circuit

As initially devised in [8] already mentioned in [3, 1], the circuit uses tracks represented by lines and quarters of circles and switches. There are three kinds of switch: the **fixed**, the **memory** and the **flip-flop** switches. They are represented by the schemes given in Fig. 1.

Note that a switch is an oriented structure: on one side, it has a single track u and, on the the other side, it has two tracks a and b. This defines two ways of crossing a switch. Call the way from u to a or b **active**. Call the other way, from a

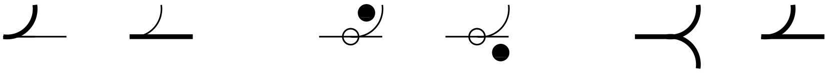


Fig. 1. The three kinds of switches. From left to right: fixed, flip-flop and memory switches.

or b to u **passive**. The names comes from the fact that in a passive way, the switch plays no role on the trajectory of the locomotive. On the contrary, in an active crossing, the switch indicates which track between a and b will be followed by the locomotive after running on u: the new track is called the **selected** track.

As indicated by its name, the **fixed switch** is left unchanged by the passage of the locomotive. It always remains in the same position: when actively crossed by the locomotive, the switch always sends it onto the same track. The flip-flop switch is assumed to be crossed only actively. Now, after each crossing by the locomotive, it changes the selected track. The memory switch can be crossed by the locomotive actively and passively. In active passage, the locomotive is sent on the selected track. Now, the selected track is defined by the track of the last passive crossing by the locomotive. Of course, at initial time, the selected track is fixed.

In [3], an infinite circuit is described which allows to simulate the working of a register machine. The global structure of the circuit contains the following parts. Two infinite periodic parts implement the two registers of the simulated machine M. Then, a finite part of the circuit implements a sequencer which controls the switching from an instruction of the programme of M to the next one. Now, in order to guarantee the return of the locomotive to the right instruction after an operation on one register, there is a selector in between the sequencer and the entry to the registers.

It is not needed to give further details on the implementation of [3] which is performed in the Euclidean plane.

3 Implementation in the hyperbolic plane

Hyperbolic geometry appeared in the first half of the 19^{th} century, in the last attempts to prove the famous parallel axiom of Euclid's *Elements* from the remaining ones. Independently, Lobachevsky and Bolyai discovered a new geometry by assuming that in the plane, from a point out of a given line, there are at least two lines which are parallel to the given line. Later, models were found, in particular Poincaré's model, which is the frame of all this study.

In this model, the hyperbolic plane is the set of points which lie in the open unit disc of the Euclidean plane whose border is the unit circle. The lines of the hyperbolic plane in Poincaré's disc model are either the trace of diametral lines or the trace of circles which are orthogonal to the unit circle, see Fig. 2. We say that the considered lines or circles **support** the hyperbolic line, h-**line** for short and, most often, simply **line** when there is no ambiguity.

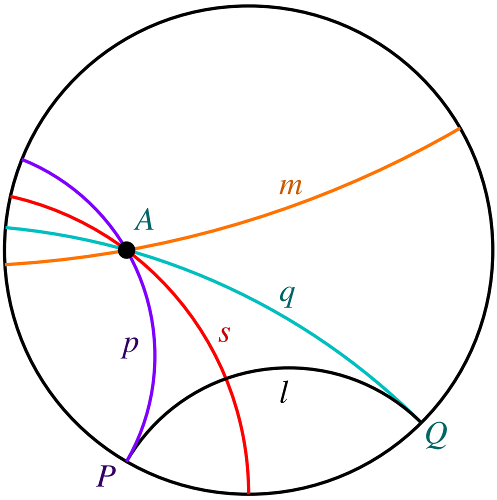


Fig. 2. The lines p and q are parallel to the line ℓ, with points at infinity P and Q, on the border of the unit disc. The h-line m is non-secant with ℓ.

The angle between two h-lines are defined as the Euclidean angle between the tangents to their support. This is one reason for choosing this model: hyperbolic angles between h-lines are, in a natural way, the Euclidean angle between the corresponding supports. In particular, orthogonal circles support perpendicular h-lines.

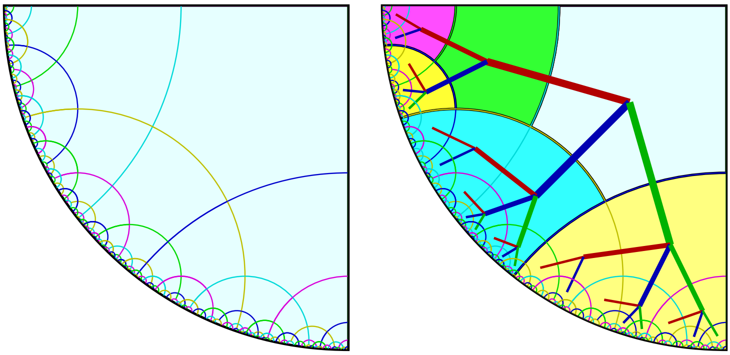
As illustrated by Fig. 2, in the hyperbolic plane, by a point A out of a line ℓ, there are exactly two lines which pass through A and out of a line ℓ, there are exactly two lines which pass through A and which are parallel to ℓ: they meet on the border of the unit disc only, the set of points at infinity which do not belong to the hyperbolic plane. There are also infinitely many ones which pass through A but which do not cut ℓ, neither in the unit disc nor outside it. Such lines are said **non-secant** with ℓ.

A striking property of this geometry is that there is no rectangle. From this, one can prove that two lines of the hyperbolic plane are non-secant if and only if they have a common perpendicular.

3.1 The pentagrid

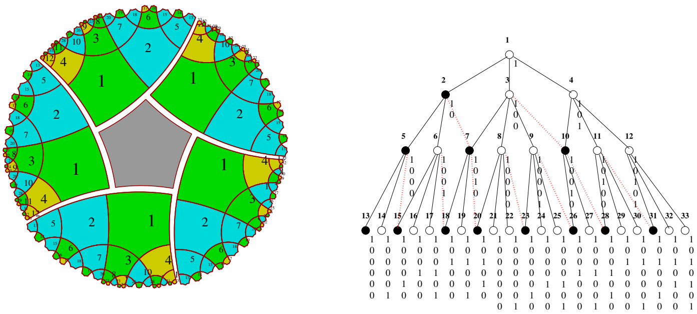
Contrary to the Euclidean plane were there are only three kinds of tilings based on the recursive replication of a regular polygon by reflection in its sides and of the images in their sides, here there are infinitely many such tilings. In one of them, the regular polygon is a pentagon with right angles. This tiling is called the **pentagrid**, see Fig. 3 and 4 for an illustrative representation. Here, we give a rough explanation of these objects, referring to [5] and to [4] for more details and for more references.

Figure 3 sketchily remembers that the tiling is spanned by a generating tree. Now, as indicated in Fig. 4, five quarters around a central tile allows us to exactly cover the hyperbolic plane with the **pentagrid** which is the tessellation obtained from the regular pentagon with right angles.



(a) (b)

Fig. 3. (a) the tiling; (b) on the right: the underlying tree which spans the tiling.



(a) (b)

Fig. 4. (a) five quarters around a central tile; (b) on the right: the representations of the numbers attached to the nodes of the Fibonacci tree.

In the right-hand side picture of Fig. 4, we remember the basic process which defines the coordinates in a quarter of the pentagrid, see [5]. We number the nodes of the tree, starting from the root and going on, level by level and, on each level, from the left to the right. Then, we represent each number in the basis defined by the Fibonacci sequence with $f_1 = 1$, $f_2 = 2$, taking the maximal representation, see[2, 5].

The main reason of this system of coordinates is that from any cell, we can find out the coordinates of its neighbours in linear time with respect to the coordinate of the cell. Also in linear time from the coordinate of the cell, we can compute the path which goes from the central cell to the cell.

Now, as the system coordinate is fixed, we can turn to the application to the implementation of cellular automata on the pentagrid.

3.2 Cellular automata on the pentagrid

A cellular automaton on the pentagrid is defined by a **local transition function** which can be put in form of a table. Each row of the table defines a **rule** and the table has seven columns numbered from 0 to 6, each entry of the table containing a state of the automaton. On each row, column 0 contains the state of the cell to which the rule applies. The rule applies because columns 1 to 5 contain the states of the neighbours of the cell defined in the following way. For the central cell, its neighbour 1 is fixed once and for all. Then, the others are numbered increasingly while counter-clockwise turning around the cell. For another cell, its neighbour 1 is its father and the other neighbours are also increasingly numbered from 2 to 5 while counter-clockwise turning around the cell.

The representation mentioned in Subsect. 3.1 allows to find the coordinates of the neighbours from that of the cell fast: in linear time from the coordinate. The list of states on a row, from column 0 to 5 is called the **context** of a rule. It is required that two different rules have different contexts. We say that the cellular automaton is **deterministic**. As there is a single row to which a rule can be applied to a given cell, the state of column 6 defines the new state of the cell. The local transition function is the function which transforms the state of a cell into its new one, also depending on the states of the neighbours as just mentioned.

An important case in the study of cellular automata is what are called **rotation invariant** cellular automata. To define this notion, we consider the following transformation on the rules. Say that the context of a rule is the **rotated image** of another one if and only if both contexts have the same state in column 0 and if one context is obtained from the other by a **circular** permutation on the contents of columns 1 to 5. Now, a cellular automaton is **rotation invariant** if and only if its table of transition T possesses the following properties:

- for each row ρ of T, T also contains four rules exactly whose contexts are the rotated image of that of ρ;
- if ρ_1 and ρ_2 are two rules of T whose contexts are the rotated image of each other, then their column 6 contain the same state.

The name of rotation invariance comes from the fact that a rotation around a tile T leaving the pentagrid globally invariant is characterized by a circular permutation on the neighbours of T defined as above.

Note that the universal cellular automaton devised in [1] is rotation invariant while the one o [7] is not. For the question of rotation invariance for cellular automata on the pentagrid, we refer the reader to [6].

Now, we can turn to the simulation of the railway circuit by a cellular automaton.

3.3 The implementation of the railway circuit

In [1], the various elements of the circuit mentioned in [3] are implemented. In fact, the paper does not give an exact description of the implementation: it only gives the guidelines, but with enough details, so that an exact implementation is useless. In this paper, we take the same implementation exactly. This is why we do not repeat it in this paper and we refer the reader to [1] for further details.

Here, we just mention the general principle of motion of the locomotive which is again taken in the new setting which we define in the next sub-section.

The tracks are implemented by portions of paths between two switches. Such a portion consists in a finite sequence of tiles which are called **blue** ones. We require that each blue tile which is not in contact with a switch has two blue neighbours exactly, where a neighbour of a tile T is another tile sharing an edge with T.

From now on, we call tiles **cells**. Most cells are in a quiescent state which we represent by the **white** colour. In the following figures of the paper, this cell is represented by a light blue colour and it is not marked by a letter, contrary to the cells which are under another state. The state of the cells which are on a blue tile is also **blue**, unless the locomotive is present on the cell. The locomotive is represented by two contiguous cells on a path. One cell is **green**, also represented by the letter G in the figures, the other is **red**, represented by the letter R. Imagine that the green cell represents the front of the locomotive and that the red cell represents its rear.

Now, the motion of the locomotive on a track is represented by Fig. 5. The easy rules can be found in [1].

Figure 5 is a space-time diagram of the evolution of the states of the cells which are along a track.

In the pentagrid, the rules become a bit more complex. We shall require that in any situation where a blue cell has two blue neighbours and three white ones exactly, it remains blue. If the cell has one green neighbour, a blue one and again three white ones, it becomes green. If a green cell has a red neighbour, a blue one and three white ones it becomes red. If a red cell has a green neighbour, a blue one and three white ones, it becomes blue. At last, a blue cell which has one red neighbour, a blue one and three white ones remains blue. Any configuration which respects these constraints defines a rule of the cellular automaton accordingly. We call these rules the **rules of the basic motion**.

The switches organize the meeting of tracks according to the principles defined in the previous section. By definition, four paths meet at a crossing and three ones meet at a switch. In [1], such a meeting is implemented by giving a special role to a cell which is the **centre** of meeting. This cell has exactly one

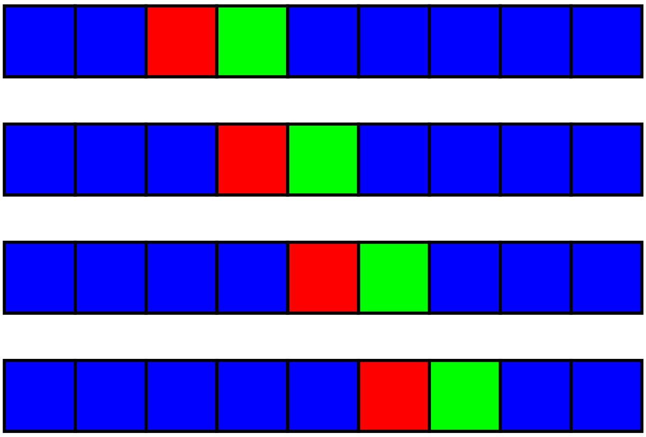


Fig. 5. The motion of the locomotive on its tracks: first approximation.

neighbour on each track: four ones in a crossing and three ones in a switch. In [1], each meeting was characterized by a special colour attached to its centre. Also, an additional tile, neighbouring the two passive tracks of a switch is also used.

In this paper, we have a different implementation which allows to reduce the number of states.

4 A new cellular automaton

In this section, we prove the following:

Theorem 1 – *There is a cellular automaton on the pentagrid which is universal and which has nine states. Moreover, the rules of the cellular automaton are rotation invariant and the cellular automaton has an infinite initial configuration which is ultimately periodic along two different rays r_1 and r_2 of the pentagrid and finite in the complement of the parts attached to r_1 and r_2.*

The idea of our implementation is to keep as much as possible to the basic motion also during the passing of a crossing or of a switch.

The configurations at a crossing and at the different kinds of switches are given in Fig. 6, 7, 8, 9 and 10. Say that such a configuration is **stable** if the locomotive is not near its centre. The stable configurations are given by the picture (g) of these figures when restricting it to the first three levels of the Fibonacci tree spanning each quarter. And so, contrarily to what was done in [1], the centre of a stable configuration is always a blue cell. Moreover, we shall stick as much as possible to the rules of the basic motion when the locomotive crosses the configuration.

We shall see that the rules which we shall devise will lead us to tune a bit the basic motion along a simple track. We shall see this later.

First, we shall look at the crossings, then at the switches in the following order: fixed, memory and flip-flop switches.

4.1 The crossings

We define the crossing as the configuration which is represented by the picture (g) of Fig. 6 or 7. These pictures introduce another simplification with respect to [1]: we assume that the paths come to the centre along the rightmost branch of a quarter spanned by a Fibonacci tree, considering that these Fibonacci trees are dispatched around the centre of the configuration. To fix things, we consider that quarter 1 is lead by the blue cell B of the stable configuration which is above the centre. There is such a leading cell, also blue and marked with B below the centre: it leads quarter 3.

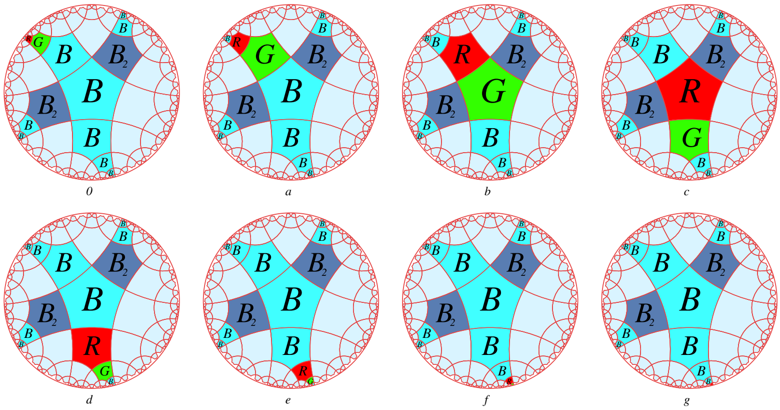


Fig. 6. The locomotive goes through a crossing: here from quarter 1 to quarter 3.

At the crossing, two paths meet. One of them goes from quarter 1 to quarter 3 along the tiles:

$$33(1),\ 12(1),\ 4(1),\ 1(1),\ 0,\ 1(3),\ 4(3),\ 12(3),\ 33(3)$$

and the others goes from quarter 5 to quarter 2 along the tiles:

$$33(5),\ 12(5),\ 4(5),\ 1(5),\ 0,\ 1(2),\ 4(2),\ 12(2),\ 33(2)$$

where a cell is indicated by its address of the form $\nu(s)$, ν being the number of the cell in its quarter and s being the number of the quarter.

Note that on the second path, from quarter 5 to 2, there are two cells with a dark blue colour marked by B_2. The role of these cells is to identify the second path.

Without loss of generality, we may assume that all crossings are rotated images of the configuration described by pictures (g) of Fig. 6 and 7. Note that

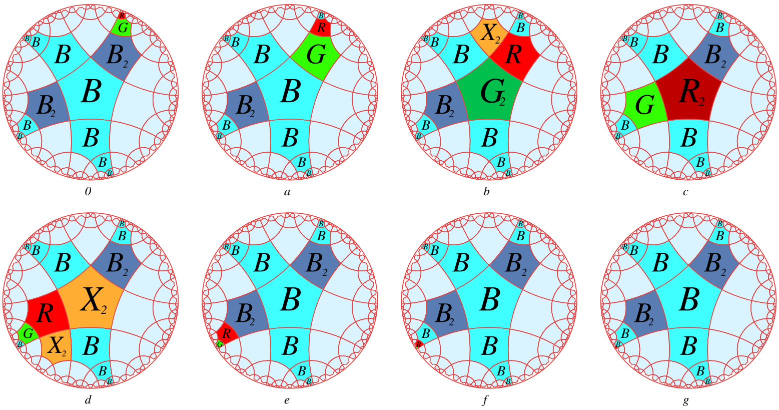


Fig. 7. The locomotive goes through a crossing: here from quarter 5 to quarter 2.

we only require the above coordinates until the third level of the Fibonacci tree spanning the corresponding quarters. On the complement of a ball of radius 4 around the centre of the crossing, the four paths may be very different. In particular, from the assumption of rotation invariance of the configurations, we may always number the quarters by counter-clockwise turning around the centre in such a way that the paths marked by B_2 are in quarters 5 and 2.

Note that the motion along the path without B_2 is simpler than the other. The basic motion rules are extended in such a way that the same colours are used by the locomotive during the crossing as the regular succession indicated by Fig. 5.

4.2 The switches

All switches are built according a similar pattern. Three tracks meet at a centre, the central cell of Fig. 8, 9 and 10. Here also, the track arriving to the switch goes along the rightmost branch of the Fibonacci tree spanning the corresponding quarter, passing through cells 33, 13, 4 and 1. Moreover, the unique arriving track is in quarter 3, and the two tracks emanating from the switch are in quarters 1 and 5. On the figures, the selected path is in quarter 1. Of course, symmetric figures where the selected paths is in quarter 5 are also possible. Now, any switch can be considered as a rotated image of the switches represented in Fig. 8, 9 and 10 and their symmetrical counterparts. Now, for all switches, cell 1(3) has two neighbours permanently in the state R_2: 2(3) and 2(4). Now, cell 2(1) characterizes the switch: it is W for the fixed switch, B_2 for the memory and the flip-flop switches.

Fixed switch The fixed switch uses the rules of the crossings too. When the locomotive arrives from track 1 or track 3, the same rules as for passing through a crossing from track 1 to track 3. It is the same in the reverse direction. Note that here, the state B_2 placed in cell 1(5) forces the locomotive to go to cell 1(1).

Figure 8 illustrates a crossing of the switch when the locomotive arrives through track 5. It is the track which is not selected by the switch. However, as passive crossings are permitted for fixed switch, the locomotive must be given the passage. Now, there is a problem: whatever the state chosen to represent the front of the locomotive in cell 0, the blue cells 1(1) and 1(3) both attract the locomotive. And so, one passage must be forbidden while the other must be forced. Note that the dark red signals at cells 2(3) and 2(4) solve the problem for attracting the locomotive to cell 1(3), but they cannot prevent it to also go to cell 1(1). This means that if nothing is decided, the switch gives rives to an additional locomotive. As the simulation requires a single one, this must be avoided. This is why when the front of the locomotive arrives at cell 1(5), it triggers a change of state in cell 2(1) which prevents the application of the basic motion rule at cell 1(1). As cell 1(1) is a neighbour of cell 2(1), it can see the new state and then understand that another rule has to be applied, forbidding the front of the locomotive to pass.

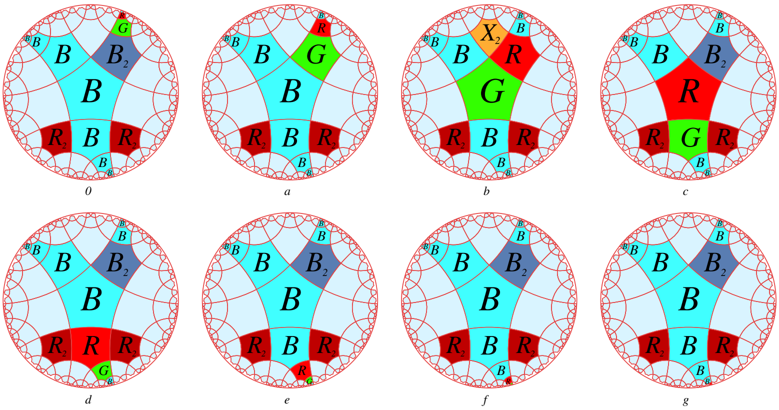


Fig. 8. The locomotive crosses a fixed switch from the non-selected track, here in quarter 5.

Now, as the same rules are considered at every cell, this changes a bit what was planned for the crossing and, the same signal as for the fixed switch occurs when the front of the locomotive comes to the crossing from track 5. This is what can be checked on Fig. 7; see picture *a* and *b* of the figure. Now, note that the same signal occurs also on picture *d*, because the configuration of cells 1(2) and 1(3) in picture *c* is the same as the configuration of cells 1(5) and 1(1) respectively in picture *a*.

This is an important point: it shows that rules devised for a certain goal may have side effects which have to be controlled.

Memory switch The memory switch is the most complicate item among the three kinds of switches.

The crossing from track 1 or 3 is not very difficult and it mainly uses the rules of a fixed switch. However, specific rules are also in use due to the state B_2 which is permanently present in cell 2(1).

Note that the memory switch is characterized by the configuration defined by cells 1(1), 2(1) and 1(5). One of cells 1(1) and 1(5) has the same state as cell (1): it is the first cell on the track which is not selected by the switch. It constitutes a **mark** which changes after the visit of the locomotive from this track.

Figure 9 illustrates the working of the switch when the locomotive comes from the non-selected track, here track 5. As the colours of the illustrations have been placed on the pentagrid from the execution of the cellular automaton by a computer program, the representation is exact. We can see that the mark is changed. The change occurs from picture *c* to picture *d* of the figure. It can take place when the rear of the locomotive is at the centre of the switch. At this moment, both cells 1(1) and 1(5) can see this state in cell 0 at the same time which allows the mark to switch from one side to the other. Note that another cell, 4(5), is temporary affected by the visit of the locomotive.

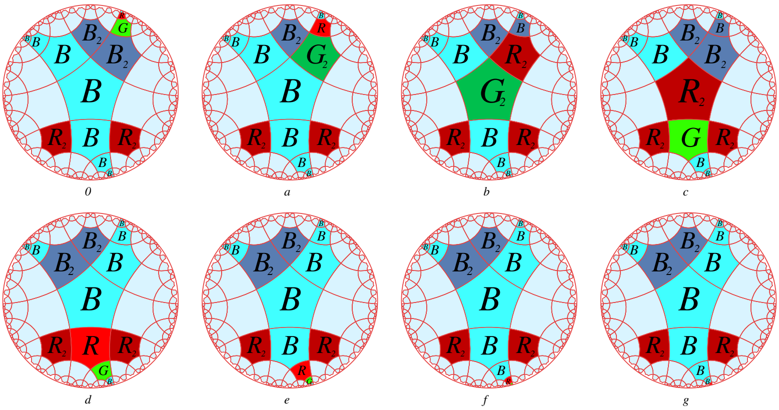


Fig. 9. The locomotive crosses a memory switch from the non-selected track, here in quarter 5. Note the change of the selected track when the rear of the locomotive leaves the switch.

Flip-flop switch With the flip-flop switch, we have a similar example of complex working. This is mainly due to the side effects of previous rules.

Here, the switch is again recognized by the configuration of cells 1(1), 2(1) and 1(5). Cell 2(1) is still in the state B_2, at least in the stable configuration. Now, cell 1(5), forbidding the access to the non-selected track, here track 5, is

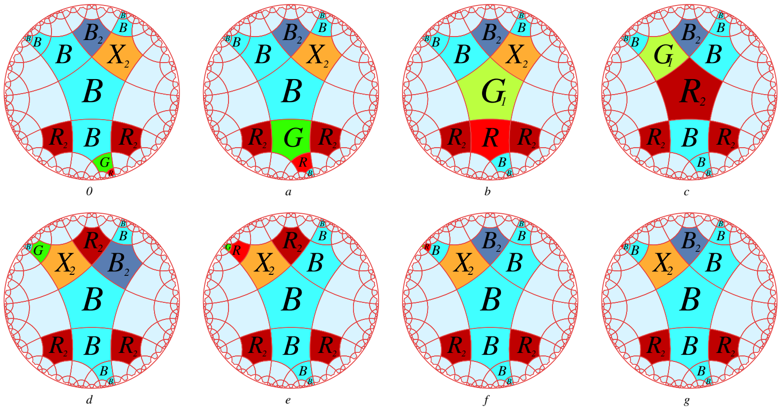


Fig. 10. The locomotive crosses a flip-flop switch, necessarily actively, here from quarter 3. Note the change of the selected track when the rear of the locomotive leaves the switch.

in the state X_2. This association of states B_2 and X_2 constitutes the mark of the flip-flop switch. When the switch changes the selected track to the other side, the mark is also changed: the state X_2 goes to the cell which is on the new non-selected track.

On Fig. 10, we can see how the change of marks occurs. We can also see that the state X_2 is used to mark the rear of the locomotive for one step of the computation. We also can see that cell 2(1) changes the state B_2 to the state R_2 for two steps. This allows to control the restoration of the red rear of the locomotive and the change of the selected track. We can notice that in picture d, cell 1(1) is in the state X_2 while cell 1(5) is in the state B_2 in place of the state B. This is why the state R_2 is maintained in cell 2(1), in order to allow the restoration of the state B in cell 1(5).

Coming back to the tracks Now, the mechanism which is used in the fixed switch has also a consequence on the tracks, on the part of them in between successive switches. Indeed, it may happen that the track has points at which there is a right angle. This means that there are three tiles sharing a vertex. We have seen that if the tiles are in the configuration of cells 1(1), 2(1) and 1(5), if the front of the locomotive is in cell 1(5), then, at the next time, cell 2(1) takes the state X_2 which blocks the advance of the locomotive. In this case, the front of the locomotive is replaced by its rear which vanishes at the next step.

A solution consists in requiring that if three cells of the track share a vertex V, then the fourth cell of the pentagrid also sharing V takes the state R_2 permanently. This allows to devise new rules in which the basic motion occurs, regardless of other possible neighbours of the cell.

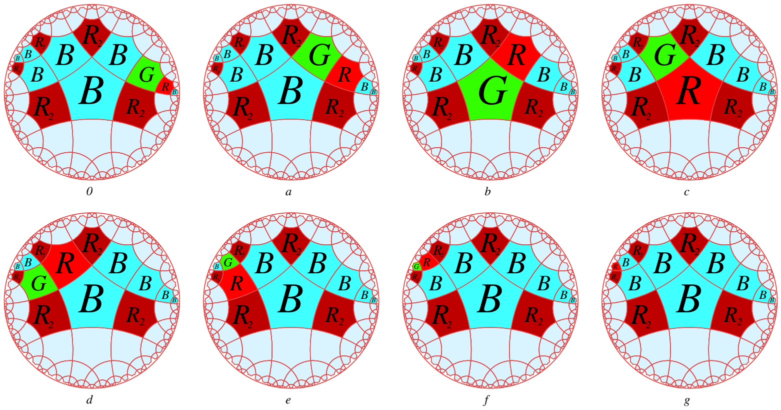


Fig. 11. The locomotive on a single track: from the right to the left.

The implementation of this solution is illustrated in Fig. 11 and 12. We may notice that both figures illustrate the same track but that the motion of the locomotive is opposite to the one of the other figure. Then, as we can see, the basic motion can be proceeded without problem.

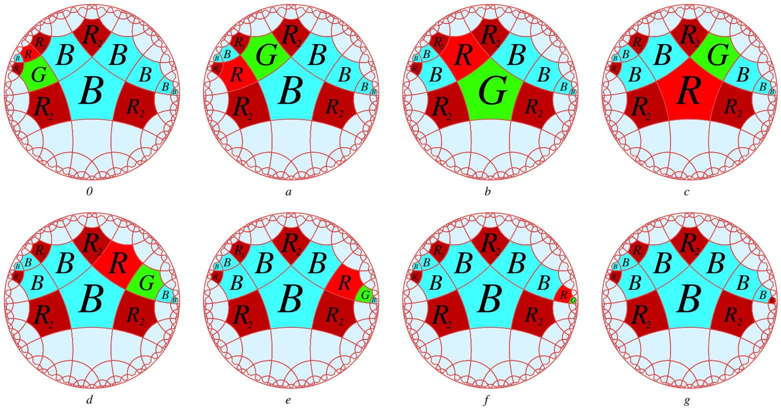


Fig. 12. The locomotive on a single track: from the left to the right.

5 The rules

As we require a rotation invariant table for our cellular automaton, it is enough to test the rules on the configurations defined by Fig. 6 to 10 of the paper and a few ones obtained by considering the switches where the selected track is track 5.

The rules of the automaton where computed by a program written in *ADA*. The program runs in an interactive way. A file in simple text contains rules to be tested. Initially, it is enough to contain the rule for the quiescent cell: `W W W W W W W,` where the states are given in the order of the table, from columns 0 to 6.

The program contains the initial configuration of the crossings and of the switches in a table 0. It contains a copy of table with no state in the cells which we call table 1. Then, we run the program. It reads the file of the rules. Starting from the central cell the program scans the quarters one after the other and in each one, from the root to the last level, level by level. For each scanned cell c, the program takes the context κ of c in table 0. Then, it compares κ with the contexts of the rules of the file. If it finds a match, it copies the state of the rule at the address of c in table 1. If it does not find a match, it asks for a new rule which the user writes into the file. Then the program is run again and the program looks at the new rule ρ: it compares it with the previous ones. If the context of ρ is different from the already stored rules, it accepts the rule. If the context matches an existing context, it looks whether the last state is the same in ρ and the matching rule. If yes, it does not accepts the rule: it already contains it. If not, then there is a contradiction and the, it displays the rule and indicates that it is contradictory with a rule of the file. In both cases, the program asks a new rule.

This process is in action until a satisfactory set of rules is obtained which runs without error starting from the initial configuration and for the time initially indicated by the user.

The program also contributes to check that the rules are rotation invariant. For this, it extracts the **reduced form** of the rules. The reduced form of a rule ρ is a word whose letters are exactly the states of ρ, given in a once and for all fixed order and with the number of occurrences of the state in ρ as an exponent, when the number is bigger than 1. Then, the rules are classified by the lexicographic order of the reduced forms. In each class, it is enough to look at the rules themselves. We first look at the rules with the same state 0. If for two such rules states 6 also coincide, the rotated rules can be appended. If they do not coincide, we have to look closer. In this case, the two contexts can be derived from each other by a permutation. If it is not a circular permutation, then again, the rotated rules can be appended. If the permutation is a circular permutation then, as states 0 are the same but not states 6, this sub-set of rules is not rotation invariant.

Thanks to the program, we have checked that the rules are rotation invariant. Table 1 gives the list of the rules. As there are 299 of them, we have given them in their encoded form as a word in $\{1..9\}^7$, these numbers being those given by the program to the states of the automaton. According to this order, the states are B, W, B_2, G, R, G_2, X_2, R_2 and G_1.

This completes the proof of theorem 1. ■

6 Conclusion

It is for sure possible to do better. However, it will require additional work to reduce the number of states by one or two of them. To go further would probably require another model of simulation.

Acknowledgement

The authors are very thankful to Professor Andrew Adamatzky for his interest to this work.

References

[1] F. Herrmann, M. Margenstern, A universal cellular automaton in the hyperbolic plane, *Theoretical Computer Science*, (2003), **296**, 327-364.

[2] M. Margenstern, New Tools for Cellular Automata of the Hyperbolic Plane, *Journal of Universal Computer Science* **6**(12), (2000), 1226–1252.

[3] M. Margenstern, Two railway circuits: a universal circuit and an NP-difficult one, Computer Science Journal of Moldova, **9**, 1–35, (2001).

[4] M. Margenstern, Cellular Automata and Combinatoric Tilings in Hyperbolic Spaces, a survey, *Lecture Notes in Computer Sciences*, **2731**, (2003), 48-72.

[5] M. Margenstern, Cellular Automata in Hyperbolic Spaces, Volume 1, Theory, *OCP*, Philadelphia, (2007), 422p.

[6] M. Margenstern, On a characterization of cellular automata in tilings of the hyperbolic plane, **ACMC'2007**, (2007)

[7] M. Margenstern, K. Morita, NP problems are tractable in the space of cellular automata in the hyperbolic plane, *Theoretical Computer Science*, **259**, 99–128, (2001)

[8] I. Stewart, A Subway Named Turing, Mathematical Recreations in *Scientific American*, (1994), 90-92.

Table 1 *Table of the codes of the rules. The rules are encoded as follows:*

1	2	3	4	5	6	7	8	9
B	W	B2	G	R	G2	X2	R2	G1

For the switches, `L` *indicates the case when the non-selected track is on the left-hand side.*

Rules for the crossing:

track 1

1 : 2222222
2 : 1131231
3 : 1122121
4 : 2132222
5 : 2122222
6 : 1122424
7 : 2212222
8 : 4122525
9 : 5422121
10 : 2422222
11 : 3122123
12 : 2312222
13 : 2322222
14 : 1322121
15 : 2242222
16 : 2522222
17 : 1522121
18 : 2252222
19 : 3122223
20 : 1431234
21 : 2432222
22 : 2342222
23 : 4531235
24 : 2532222
25 : 3422123
26 : 2352222
27 : 1422124
28 : 5134231
29 : 3522123
30 : 4522125
31 : 1135231

track 5

32 : 5122421
33 : 1122521
34 : 1322424
35 : 3122424
36 : 4322525
37 : 1131246
38 : 2142227
39 : 6131258
40 : 1672121
41 : 7152222
42 : 2712222
43 : 2722222
44 : 2272222
45 : 3622124
46 : 1622121
47 : 2622222
48 : 5622173
49 : 8141237
50 : 1822121
51 : 4822125
52 : 2412222
53 : 2822222
54 : 3822123
55 : 7151231
56 : 1722121
57 : 5722473
58 : 2512222
59 : 1772121
60 : 2742222
61 : 3722123
62 : 3122523

Rules for the crossing (continuation):

track 3

63 : 5322421
64 : 1322521
65 : 1134234
66 : 4135235

track 2

67 : 5431231
68 : 1531231
69 : 1141236
70 : 6151238

Rules for the fixed switches:

track 1

71 : 8131247
72 : 7131251
73 : 1121231
74 : 2122282
75 : 1182181
76 : 8122228
77 : 2282222
78 : 2182222
79 : 8212228
80 : 2812222
81 : 1421234
82 : 4521235
83 : 2422282
84 : 1482184
85 : 2482222
86 : 5124231
87 : 2522282
88 : 4582185
89 : 8422228
90 : 2582222
91 : 8242228
92 : 1125231
93 : 5182481
94 : 8522228
95 : 8252228
96 : 2842222

track 5

97 : 1182581
98 : 2852222
99 : 1121244
100 : 4121255

track 3

101 : 1472121
102 : 5422173
103 : 1182484
104 : 1124234
105 : 4182585
106 : 4125235
107 : 5482181
108 : 5421231

track 5, L

109 : 1582181
110 : 1521231
111 : 1321211
112 : 2232222
113 : 1321244
114 : 4321255

track 3, L

115 : 5324211
116 : 1325211
117 : 1324214
118 : 4325215

track 1, L

119 : 5321241
120 : 1321251
121 : 1421216
122 : 6521218
123 : 5622123
124 : 2622282
125 : 1682184
126 : 2682222
127 : 8324211
128 : 2822282
129 : 4882185

Rules for the memory switches:

track 1

130 : 2882222
131 : 1132121
132 : 3132223
133 : 3122133
134 : 1132424
135 : 4132525
136 : 3432223
137 : 5432121
138 : 3532223
139 : 3422133
140 : 1532121

track 5

141 : 3522133
142 : 3122436
143 : 1121266
144 : 3162223
145 : 6122538
146 : 6121288
147 : 1632121
148 : 3182223
149 : 2332222
150 : 8622333
151 : 8124231
152 : 1832123
153 : 3822331
154 : 3322121
155 : 3132123
156 : 3312223

track 3

157 : 1122131
158 : 1432124

track 5, L

159 : 4532125
160 : 5132421
161 : 1132521
162 : 1122434
163 : 3342223
164 : 4122535
165 : 3432123
166 : 3352223
167 : 5422131
168 : 3532123

track 1, L

169 : 1522131
170 : 3132426
171 : 1621216
172 : 6132528
173 : 3612223
174 : 2262222
175 : 6821218
176 : 8632323
177 : 3812223
178 : 1622131
179 : 3832321

track 3, L

180 : 1822133
181 : 1422134
182 : 4522135
183 : 5122431

Rules for the flip-flop switches:

```
184 : 1122531
185 : 1121271
186 : 3172223
187 : 7122137
188 : 1124279
189 : 9125278
190 : 1932129
191 : 2922282
192 : 5982181
193 : 2982222
194 : 7922131
195 : 8921211
196 : 9832127
197 : 3912228
198 : 2922222
```

```
199 : 1922124
200 : 2292222
201 : 1882181
202 : 1721231
203 : 7182427
204 : 8732228
205 : 4722125
206 : 3122181
207 : 1721211
208 : 7182527
209 : 8712223
210 : 5722421
211 : 1122181
212 : 7132127
213 : 3712223
```

```
L
214 : 1722521
215 : 1724219
216 : 9725218
217 : 7932121
218 : 1922139
219 : 8121291
220 : 3192228
221 : 9822137
222 : 1321271
223 : 3182121
224 : 8372228
225 : 7122487
226 : 1182121
227 : 8172223
```

Rules for the tracks:

```
228 : 7122587
229 : 1182811
230 : 8112228
231 : 8122218
232 : 1811821
233 : 1182211
234 : 1812121
235 : 2822242
236 : 2222252
237 : 1112281
238 : 1184224
239 : 4125225
240 : 5421221
241 : 2222212
242 : 8122248
243 : 2822252
244 : 1142284
245 : 4185225
246 : 1521221
247 : 1182844
248 : 8142228
249 : 8122258
```

```
250 : 2822212
251 : 4152285
252 : 5481221
253 : 1121221
254 : 4182855
255 : 1482814
256 : 8152228
257 : 8412228
258 : 8422218
259 : 5412281
260 : 1581221
261 : 5482811
262 : 4582815
263 : 8512228
264 : 1841824
265 : 8522218
266 : 1512281
267 : 1181221
268 : 1582811
269 : 5182841
270 : 4851825
271 : 1482214
```

```
272 : 1182851
273 : 5814821
274 : 4582215
275 : 1842124
276 : 1815821
277 : 5182241
278 : 4852125
```

the other direction:

```
279 : 1814824
280 : 4182255
281 : 5842121
282 : 4815825
283 : 5482211
284 : 1852121
285 : 5841821
286 : 1582211
287 : 1851821
288 : 1412284
289 : 4512285
290 : 1481224
```

```
291 : 5142281
292 : 4581225
293 : 1421224
294 : 1152281
295 : 5184221
296 : 4521225
297 : 2222242
298 : 1185221
299 : 5124221
```

.

Automorphisms of transition graphs for a linear cellular automaton

Edward J. Powley and Susan Stepney

Department of Computer Science, University of York, UK

Abstract. A *cellular automaton (CA)* is a discrete dynamical system, and the *transition graph* is a representation of the CA's phase space. *Automorphisms* of the transition graph correspond to symmetries of the phase space; studying how the total number of automorphisms varies with the number of cells on which the CA operates yields a (partial) classification of the space of CA rules according to their dynamical behaviour.

In the general case, to find the number of automorphisms we must iterate over the entire transition graph; thus the time complexity is exponential with respect to the number of cells. However, if the CA is *linear*, the transition graph has properties which allow the number of automorphisms to be computed much more efficiently. In this paper, we investigate the numbers of automorphisms for a particular linear CA, *elementary rule 90*. We observe a relationship between the number of automorphisms and a number theoretical function, the *suborder function*.

1 Introduction

A *cellular automaton (CA)* consists of a finite nonempty set of *states*, a discrete lattice of *cells*, and a *local update rule* which maps deterministically the state of a cell and its neighbours at time t to the state of that cell at time $t + 1$. A *configuration* of a CA is an assignment of a state to each cell. The local update rule extends to a *global map*, a function from configurations to configurations, in the natural way.

The *transition graph* of a CA is a directed graph whose vertices are the configurations of the CA, and whose edges are determined by the global map. There is an edge from vertex r to vertex s if and only if the global map sends configuration r to configuration s. The transition graph is a representation of the overall structure of the phase space of the CA: in particular, the *automorphisms* (self-isomorphisms or "symmetries") of the transition graph are, in a sense, the symmetries of the CA's dynamics [1]. Examples of transition graphs are shown in Figs. 1 and 2.

In [1], we investigate how numbers of automorphisms vary with the number N of cells on which the CA operates (Fig. 3). For the majority of CA rules, there seems to be a linear relationship between the number of automorphisms and e^{e^N}.

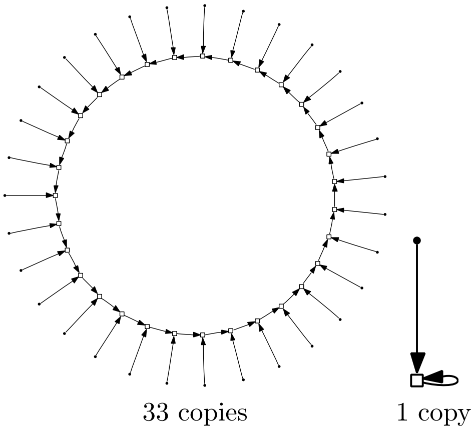


Fig. 1. Transition graph for rule 90 on 11 cells.

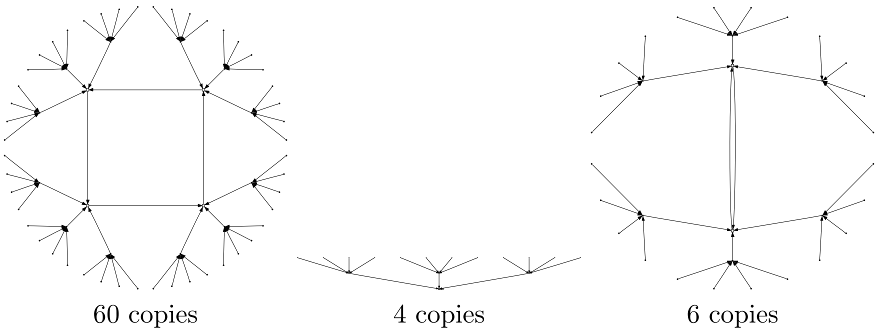


Fig. 2. Transition graph for rule 90 on 12 cells.

However, we identify two classes of CAs for which this linear correspondence does not seem to hold. One of these classes consists almost entirely of *linear CAs* (CAs whose local update rule is a linear function), and is characterised by the "zig-zag" pattern depicted in Fig. 4 (a). In this paper, we investigate this pattern more closely.

As is the case with many other classes of system, linear CAs submit much more readily to analysis than their non-linear counterparts. Indeed, the operation of a linear CA is simply repeated convolution of a configuration with a fixed sequence corresponding to the rule, which in turn is equivalent to repeated multiplication in a finite ring of polynomials. Martin et al [2] use this fact to study linear CAs, and succeed in proving several results about one linear CA in particular (*elementary rule 90*, in Wolfram's terminology [3]). We use these results to derive an algorithm, dramatically more efficient than the general al-

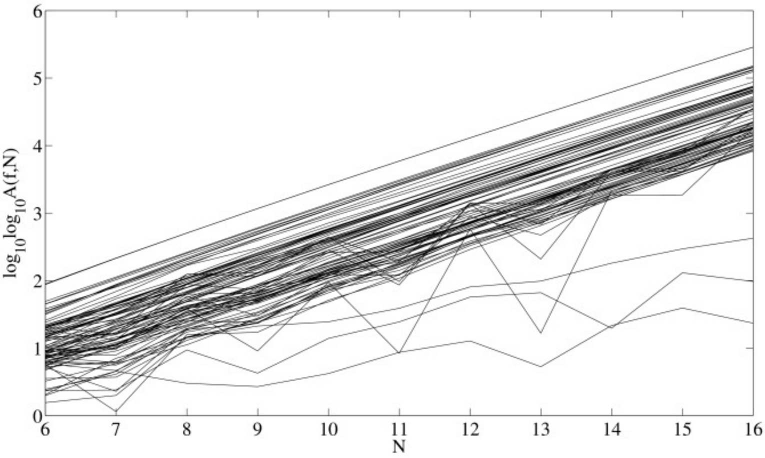


Fig. 3. Plot of $\log_{10} \log_{10} A(f, N)$ (where $A(f, N)$ is the number of automorphisms) against N, for $6 \leq N \leq 16$ and for all 88 essentially different ECA rules. From [1].

gorithm described in [1], for computing the number of automorphisms for the transition graphs of rule 90.

We argue, but do not prove, that the "zig-zag" oscillations in the number of automorphisms for rule 90 on N cells correspond to the oscillations of a number theoretical function known as the *suborder function*.

2 Linear CAs and polynomials

We restrict our attention to finite one-dimensional CAs, i.e. we take the lattice to be $\mathbb{Z}_N$ (the cyclic group of integers modulo N). This lattice has *periodic boundary condition*, in that we consider cell $N-1$ to be adjacent to cell 0. The neighbourhood is specified in terms of its *radius* r, so that the neighbours of cell i are cells $i-r, \dots, i+r$. We further restrict our attention to CAs whose state set is also a cyclic group, say $\mathbb{Z}_k$. Thus the local update rule is a function $f : \mathbb{Z}_k^{2r+1} \to \mathbb{Z}_k$, which extends to a global map $F : \mathbb{Z}_k^N \to \mathbb{Z}_k^N$.

Such a CA is said to be *linear* if the local update rule is a linear function; that is, if there exist constants $\lambda_{-r}, \dots, \lambda_r$ such that

$$f(x_{-r}, \dots, x_r) = \lambda_{-r} x_{-r} + \cdots + \lambda_r x_r \ , \tag{1}$$

where the operations of addition and multiplication are the usual operations of modular arithmetic.

Martin et al [2] study linear CAs by means of polynomials. Denote by R_k^N the set of polynomials with coefficients over $\mathbb{Z}_k$ of degree at most $N-1$. We

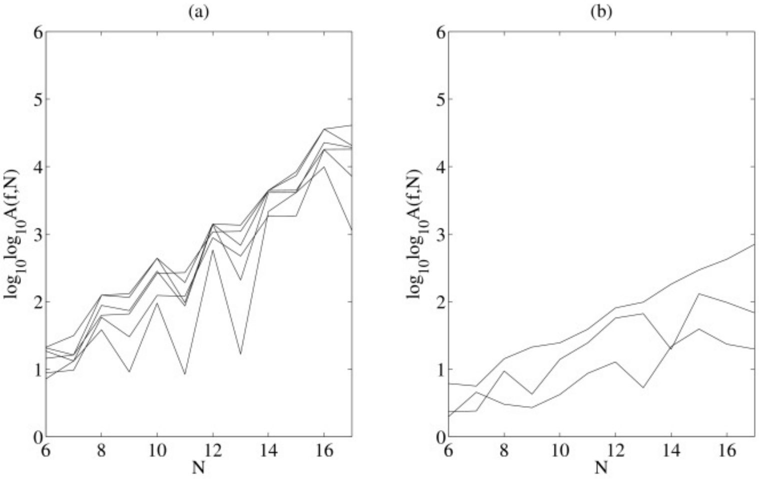


Fig. 4. As Fig. 3, but for $6 \leq N \leq 17$, and showing the two classes of ECAs which do not exhibit a linear relationship between numbers of automorphisms and e^{e^N}. From [1].

can define addition and multiplication in R_k^N similarly to the usual arithmetic of polynomials, but setting $x^N = 1$ (so in effect, powers are computed modulo N). Under these operations, R_k^N is a ring.

Let f be a local update rule of the form of Equation 1. We associate with f the polynomial T_f in R_k^N defined by

$$T_f(x) = \lambda_{-r}x^r + \cdots + \lambda_r x^{-r} . \tag{2}$$

Furthermore, we associate with a configuration $s = a_0 a_1 \ldots a_{N-1} \in \mathbb{Z}_k^N$ the polynomial

$$A_s(x) = a_0 + a_1 x + \cdots + a_{N-1}x^{N-1} . \tag{3}$$

Then the polynomial associated with the configuration $F(s)$ is simply $T_f(x)A_s(x)$. In other words, repeated application of the global map F corresponds to repeated multiplication by the polynomial $T_f(x)$.

2.1 Rule 90

Let $r = 1$ and $k = 2$, and consider $f : \mathbb{Z}_2^3 \to \mathbb{Z}_2$ defined by

$$f(x_{-1}, x_0, x_1) = x_{-1} + x_1 . \tag{4}$$

Since $r = 1$ and $k = 2$, this is an example of an *elementary cellular automaton.* According to Wolfram's numbering scheme [3], f is *rule 90.* The polynomial corresponding to f is

$$T_f(x) = x + x^{-1} . \tag{5}$$

Remark 2.1. Let F be the global map for rule 90 on N cells. Choose an initial configuration s_0, and let

$$s_t = \underbrace{F \circ \cdots \circ F}_{t \text{ occurrences}}(s_0) \; . \tag{6}$$

Then at most $O(\log t)$ polynomial multiplications are required to compute s_t.

Proof. It suffices to show that the polynomial $(T_f(x))^t$ can be written as a product of at most $O(\log t)$ polynomials, where each term in this product is either known or computable in constant time.

Since we are working with coefficients over $\mathbb{Z}_2$, we have

$$\left(x + x^{-1}\right)^{2^k} = x^{2^k} + x^{-2^k} \tag{7}$$

for all nonnegative integers k. Thus if t is a power of 2, s_t can be computed by multiplication with $x^t + x^{-t}$.

If t is not a power of 2, it can nevertheless be written as a sum of $\lceil \log_2 t \rceil$ or fewer powers of 2 (i.e. in binary notation). If

$$t = 2^{i_1} + \cdots + 2^{i_l} \; , \tag{8}$$

then

$$\left(x + x^{-1}\right)^t = \left(x + x^{-1}\right)^{2^{i_1}} \ldots \left(x + x^{-1}\right)^{2^{i_l}} \; . \tag{9}$$

The product on the right-hand side involves no more than $\lceil \log_2 t \rceil$ terms, each of which can be determined in constant time via Equation 7. □

2.2 Cycle lengths and the suborder function

For positive integers n and k, the *(multiplicative) suborder function* $\text{sord}_n(k)$ is defined [2] by

$$\text{sord}_n(k) = \begin{cases} \min\left\{j > 0 \;:\; k^j \equiv \pm 1 \mod n\right\} & \text{if such a } j \text{ exists} \\ 0 & \text{otherwise} \; . \end{cases} \tag{10}$$

Note that $\text{sord}_n(k) \neq 0$ if and only if n and k are relatively prime. In particular, if $k = 2$ then $\text{sord}_n(2)$ is nonzero if n is odd and zero if n is even. The suborder function $\text{sord}_n(2)$ is plotted in Fig. 5.

If n is odd, then we have

$$\log_2 n \leq \text{sord}_n(2) \leq \frac{n-1}{2} \; . \tag{11}$$

The set of values of n for which the upper bound is achieved is a subset of the primes.

Let Π_N denote the length of the cycle reached by rule 90 from an initial configuration which assigns state 1 to a single cell and state 0 to the remainder.

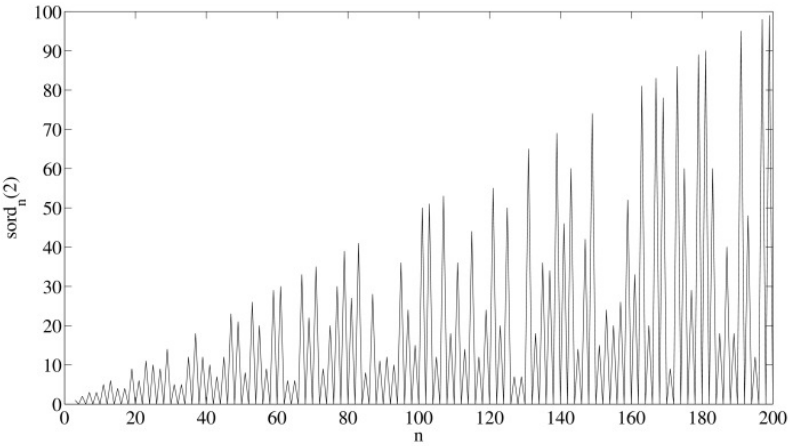


Fig. 5. Plot of the suborder function $\mathrm{sord}_n(2)$ against n, for $3 \leq n \leq 200$.

Due to rule 90's linearity, all cycle lengths must be factors of Π_N. Furthermore, Martin et al [2] show that

$$\Pi_N = \begin{cases} 1 & \text{if } N \text{ is a power of 2} \\ 2\Pi_{N/2} & \text{if } N \text{ is even but not a power of 2} \\ \text{a factor of } 2^{\mathrm{sord}_N(2)} - 1 & \text{if } N \text{ is odd .} \end{cases} \tag{12}$$

3 Counting automorphisms of transition graphs

Definition 3.1. *Consider a CA whose set of configurations is C and whose global map is F. The* transition graph *for this CA is the directed graph with vertex set C and edge set*

$$\{(s, F(s)) \ : \ s \in C\} \ . \tag{13}$$

Every vertex in a transition graph has out-degree 1. This forces the graph to have a "circles of trees" topology: the graph consists of a number of disjoint cycles, with a (possibly single-vertex) tree rooted at each vertex in each cycle.

The *basins* of the transition graph are its disjoint components: each basin consists of exactly one cycle, along with the trees rooted on that cycle.

Examples of transition graphs are shown in Figs. 1 and 2.

Definition 3.2. *Consider a directed graph with vertex set V and edge set E. An* automorphism *on this graph is an isomorphism from the graph to itself; in other words, a bijection $\alpha : V \to V$ such that*

$$(x, y) \in E \iff (\alpha(x), \alpha(y)) \in E \ . \tag{14}$$

Denote by $A(f, N)$ the number of automorphisms in the transition graph for the CA with local rule f on N cells.

In [1] we describe an algorithm for computing the number of automorphisms for a transition graph. This algorithm works by exploiting the recursive structure of the transition graph. For instance, consider a tree, rooted at vertex r, such that the "children" of r are vertices $c_1, \dots, c_k$. Then the number of automorphisms in the tree is the product of the numbers of automorphisms for each subtree rooted at a c_i, multiplied with the number of permutations of $c_1, \dots, c_k$ which preserve the isomorphism classes of the subtrees.

Transition graphs for linear CAs have further structure to be exploited:

Lemma 3.1 ([2, Lemma 3.3]). *In a transition graph for a linear CA, the trees rooted at the vertices in the cycles form a single isomorphism class.*

Thus two basins are isomorphic if and only if their cycles have the same length. Cycles of different lengths can occur within a transition graph, so the basins do not necessarily form a single isomorphism class. To find the isomorphism classes, it is necessary (and sufficient) to find the lengths and multiplicities of the cycles. Martin et al [2] give an algorithm for this in rule 90, and it seems reasonable to expect that similar algorithms exist for other linear CAs.

The following results characterise the structure of the trees themselves for rule 90 on N cells:

Theorem 3.1 ([2, Theorem 3.3]). *If N is odd, all trees in the transition graph consist of a single edge.*

Theorem 3.2 ([2, Theorem 3.4]). *If N is even, all trees in the transition graph have the following properties:*

1. *The distance from the root vertex to every leaf vertex is*

$$\frac{1}{2} \max\left\{2^j \ : \ 2^j | N\right\}; \tag{15}$$

2. *The root vertex has in-degree 3;*
3. *Every non-root non-leaf vertex has in-degree 4.*

These theorems are illustrated in Figs. 1 and 2.

If N is odd, clearly the only automorphism on each tree is the identity. How many automorphisms does each tree possess if N is even? The following result is an application of [1, Lemma 1].

Lemma 3.2. *Consider a tree of depth $D > 1$, whose root vertex v has in-degree 3 and with all other vertices having in-degree 4. The number of automorphisms for this tree is*

$$A(v) = 24^{2^{2(D-1)}}/4\,. \tag{16}$$

Proof. See Appendix 7. □

The following theorem is our main result, and follows directly from Lemma 3.2 above and Lemma 2 and Theorem 2 in [1].

Theorem 3.3. *Suppose that, for some value of N, the distinct cycle lengths in rule 90 are $l_1, \dots, l_k$, and there are m_i cycles of length l_i. Let*

$$A_T = \begin{cases} 1 & \text{if } N \text{ is odd} \\ 24^{2^{N-2}} / 4^{2^{N-D_2(N)}} & \text{if } N \text{ is even} , \end{cases} \tag{17}$$

where

$$D_2(N) = \max\left\{2^j \ : \ 2^j | N\right\} . \tag{18}$$

Then

$$A(90, N) = \left(\prod_{i=1}^{k} m_i! \cdot l_i^{m_i} \right) \cdot A_T . \tag{19}$$

Proof. See Appendix 7. □

Thus if the l_is and m_is are known, the number of automorphisms can easily be calculated. The following corollary illustrates a particularly straightforward special case:

Corollary 3.1. *If N is a power of 2, then*

$$A(90, N) = 24^{2^{N-2}} / 4 . \tag{20}$$

Proof. See Appendix 7. □

4 Computational results

Martin et al [2] provide cycle lengths and multiplicities for $3 \leq N \leq 40$, as well as an algorithm for computing the lengths and multiplicities for larger N. Using these in conjunction with Theorem 3.3, we are able to compute values of $A(90, N)$ for N much larger than by the general method described in [1]. Some results are shown in Fig. 6.

Compare Fig. 6 with the suborder function $\mathrm{sord}_N(2)$ plotted in Fig. 5. In particular, observe that peaks in one seem to correspond with troughs in the other. Indeed, it can be verified numerically that we have an approximate linear relationship:

$$\log_{10} \log_{10} A(90, N) \approx 0.30N - 0.28 \,\mathrm{sord}_N(2) - 0.04 . \tag{21}$$

Figure 7 plots the two sides of Equation 21, and Fig. 8 plots the difference between them against N. Although the correlation is not exact, note that there are no outliers. Also note that the difference between the two sides, and hence the error in this approximation, seems to increase (albeit slowly) with N.

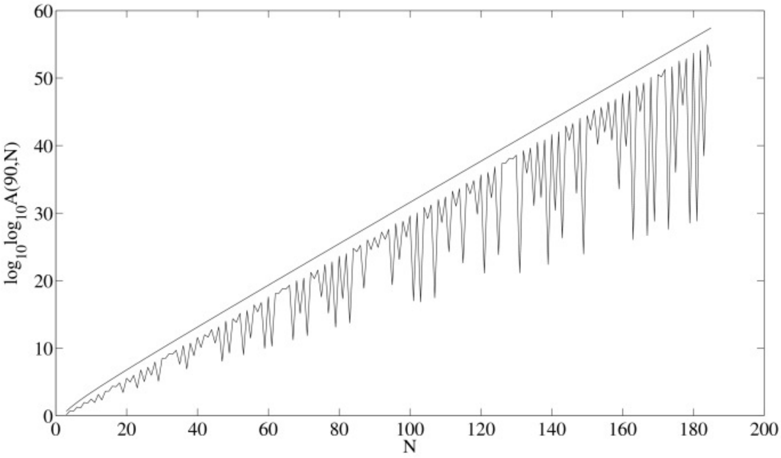


Fig. 6. Plot of $\log_{10}\log_{10} A(90, N)$ (lower line) against N, for $3 \leq N \leq 185$. For comparison, $\log_{10}\log_{10} A(204, N) = 2^N!$ is also plotted (upper line).

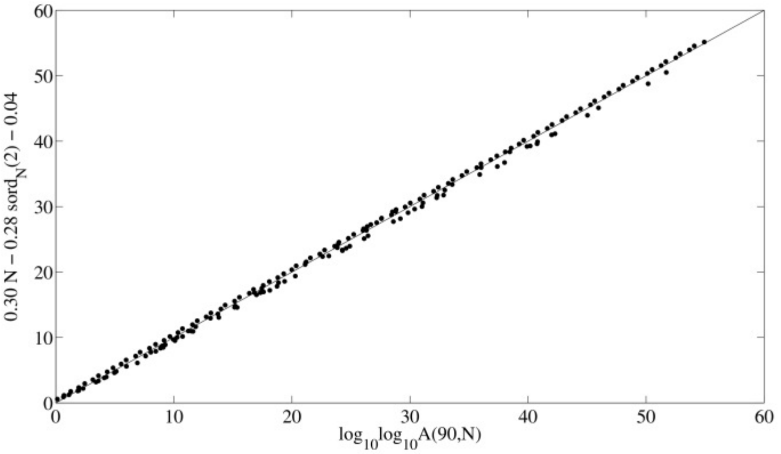


Fig. 7. Plot of the two sides of Equation 21. The diagonal "$y = x$" line is plotted for comparison.

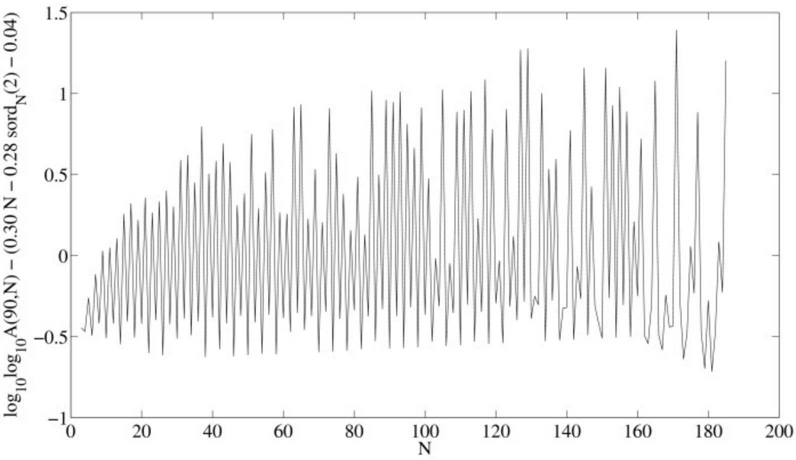


Fig. 8. Plot of the difference between the two sides of Equation 21 against N.

5 Conclusion

Previously [1] we computed numbers of automorphisms for all 88 essentially different ECAs. Implementing the "brute force" method described therein on a current desktop PC, we find that $N = 17$ is the practical limit of what can be computed in a reasonable length of time. Furthermore, the exponential complexity of the computation means that an increase in computational resources would not significantly increase this limit. In contrast, rule 90 has properties which allow for a much more efficient algorithm. On the same desktop PC, we are able to count automorphisms for $N \leq 185$, and for many (though increasingly uncommon) cases beyond, with ease.

However, it seems plausible that there exists an even simpler expression for the number of automorphisms in rule 90, and that the suborder function $\mathrm{sord}_N(2)$ dominates this expression. The suborder function relates to rule 90 since, if N is odd, all cycle lengths must divide $2^{\mathrm{sord}_N(2)} - 1$. It is not clear why the expression

$$\prod_i m_i! \cdot l_i^{m_i} \ , \tag{22}$$

where the l_is are the cycle lengths and the m_is are their respective multiplicities, should be so strongly correlated with this common multiple of the l_is. We plan to investigate this further, and to determine whether this approximate correlation does indeed indicate the existence of a simpler exact expression for $A(90, N)$.

It is reasonable to expect that other linear rules admit a similar approach to that applied here. Indeed, Martin et al [2] generalise some (but not all) of their results beyond rule 90. We intend to use these more general results to extend our methods to the other linear ECAs, and to other linear CAs in general.

However, these methods almost certainly do not apply to nonlinear CAs: the analogy with finite rings of polynomials is crucial to this work, but this analogy only holds for linear CAs. Thus this work demonstrates (if yet another demonstration were needed!) the ease of analysis and computation for linear CAs as compared to their nonlinear counterparts.

Acknowledgment

This work is funded by an EPSRC DTA award.

6 Appendix

7 Proofs

7.1 Proof of Lemma 3.2

Let u_i be any vertex at depth i in the tree, so $v = u_0$ and u_D is a leaf. Then by [1, Lemma 1], noting that the children of u_i form a single isomorphism class, we have

$$A(v) = A(u_0) = 3!A(u_1)^3 \tag{23}$$

$$A(u_1) = 4!A(u_2)^4 \tag{24}$$

$$\vdots$$

$$A(u_{D-1}) = 4!A(u_D)^4 \tag{25}$$

$$A(u_D) = 1 \; . \tag{26}$$

Thus

$$A(v) = 3!\underbrace{(4!(4!(\dots(1)^4\dots)^4)^4)^3}_{D-1 \text{ occurrences of } 4!} \tag{27}$$

$$= 3! \times 4!^3 \times 4!^{3\times 4} \times \cdots \times 4!^{3\times 4^{D-2}} \tag{28}$$

$$= 3! \times 4!^{3\sum_{i=0}^{D-2} 4^i} \; . \tag{29}$$

It can be shown that, for any positive integers n and k,

$$\sum_{i=0}^{n} k^i = \frac{k^{n+1}-1}{k-1} \; . \tag{30}$$

Thus

$$\begin{aligned} A(v) &= 3! \times 4!^{3\sum_{i=0}^{D-2} 4^i} && (31)\\ &= 3! \times 4!^{3\times(4^{D-1}-1)/3} && (32)\\ &= \frac{3!}{4!} \times 4!^{4^{D-1}} && (33)\\ &= 24^{2^{2(D-1)}}/4 && (34) \end{aligned}$$

as required. □

7.2 Proof of Theorem 3.3

[1, Theorem 2] states that

$$A(90, N) = \left(\prod_{I \in \{B_i\}/\cong} |I|! \right) \left(\prod_{i=1}^{k} A(B_i)^{m_i} \right) . \tag{35}$$

By [2, Lemma 3.3], all of the trees rooted at vertices in cycles are isomorphic. Thus two basins are isomorphic if and only if they have the same cycle length, and so we have

$$\prod_{I \in \{B_i\}/\cong} |I|! = \prod_{i=1}^{k} m_i! \, . \tag{36}$$

Now let $A(B_i)$ be the number of automorphisms for a basin whose cycle length is l_i. By [1, Lemma 2], we have

$$A(B_i) = \frac{l_i}{q} \prod_{j=1}^{l_i} A(v_j) \, . \tag{37}$$

But all of the trees are isomorphic, so $q = 1$ and thus

$$A(B_i) = l_i A(v)^{l_i} \, , \tag{38}$$

where $A(v)$ is the number of automorphisms in a tree. Substituting into Equation 35 we have

$$\begin{aligned} A(90, N) &= \left(\prod_{i=1}^{k} m_i! \right) \left(\prod_{i=1}^{k} (l_i A(v)^{l_i})^{m_i} \right) && (39)\\ &= \prod_{i=1}^{k} \left(m_i! \cdot l_i^{m_i} \cdot A(v)^{l_i m_i} \right) && (40)\\ &= \left(\prod_{i=1}^{k} m_i! \cdot l_i^{m_i} \right) \cdot A(v)^{\sum_{i=1}^{k} l_i m_i} \, . && (41) \end{aligned}$$

It now suffices to show that

$$A(v)^{\sum_{i=1}^{k} l_i m_i} = A_T \tag{42}$$

with A_T as defined in Equation 17.

If N is odd, [2, Theorem 3.3] states that all trees consist of a single edge. Thus $A(v) = 1$, and so $A(v)^{\sum_{i=1}^{k} l_i m_i} = 1 = A_T$, regardless of the values of $l_i m_i$.

Suppose that N is even. By [2, Theorem 3.4], all trees are of the form described in Lemma 3.2, with $D = D_2(N)/2$. Thus we have

$$A(v) = 24^{2^{D_2(N)-2}}/4\,. \tag{43}$$

Now, $\sum_{i=1}^{k} l_i m_i$ is simply the number of configurations which occur in cycles, and thus, by a corollary to [2, Theorem 3.4], is given by

$$\sum_{i=1}^{k} l_i m_i = 2^{N-D_2(N)}\,. \tag{44}$$

Hence

$$\begin{aligned}
A(v)^{\sum_{i=1}^{k} l_i m_i} &= 24^{2^{D_2(N)-2}\times 2^{N-D_2(N)}}/4^{2^{N-D_2(N)}} && (45)\\
&= 24^{2^{D_2(N)-2+N-D_2(N)}}/4^{2^{N-D_2(N)}} && (46)\\
&= 24^{2^{N-2}}/4^{2^{N-D_2(N)}} && (47)\\
&= A_T && (48)
\end{aligned}$$

as required. □

7.3 Proof of Corollary 3.1

If N is a power of 2 then $D_2(N) = N$, so

$$A_T = 24^{2^{N-2}}/4 \tag{49}$$

Furthermore, by [2, Lemmas 3.4 and 3.5], the only possible cycle length is 1; by [2, Lemma 3.7], there is only one such cycle. Thus

$$\begin{aligned}
A(90, N) &= \left(1! \cdot 1^1\right) \cdot A_T && (50)\\
&= A_T && (51)\\
&= 24^{2^{N-2}}/4 && (52)
\end{aligned}$$

as required. □

References

[1] Powley, E., Stepney, S.: Automorphisms of transition graphs for elementary cellular automata. In: proceedings of Automata 2007: 13th International Workshop on Cellular Automata. (2008) To appear in Journal of Cellular Automata.

[2] Martin, O., Odlyzko, A.M., Wolfram, S.: Algebraic properties of cellular automata. Communications in Mathematical Physics **93** (1984) 219–258.

[3] Wolfram, S.: Statistical mechanics of cellular automata. Reviews of Modern Physics **55**(3) (1983) 601–644.

The discrete Baker transformation

Burton Voorhees

Athabasca University, 1 University Drive, Athabasca T9S 3A3, Canada
burt@athabascau.ca

The baker transformation, or Bernoulli shift is defined for a continuous dynamical system $f : [0,1] \rightarrow [0,1]$ by $f(x) = 2x \mod (1)$. In recent papers a discrete version of this transformation has been defined, acting on strings $(a_0, \ldots, a_{n-1})$ where the a_i are contained in $\{0, \ldots, p-1\}$ with p prime. These strings define additive cellular automata acting on a cylinder of size n. (More generally, this discrete transformation is defined acting on d-dimensional arrays of numbers representing additive cellular automata on a d-dimensional torus but only dimension one is considered here.) In [1] the discrete baker transformation was used to obtain a number of results on the behavior of additive cellular automata, including strong estimates of tree heights and cycle lengths in state transition diagrams. In [2], non-singular transformations of additive cellular automata yielding rules with isomorphic state transition diagrams are considered and shown to be combinations of shifts and powers of the baker transformation. In [3] cycle properties of the discrete baker transformation itself are considered. In this paper properties of the discrete baker transformation are discussed and its use in studies of additive cellular automata is illustrated. In addition, extension to non-linear cellular automata is considered.

References

[1] Bulitko, V., Voorhees, B., and Bulitko, V. (2006) Discrete baker transformations for linear cellular automata analysis. Journal of Cellular Automata 1(1) 40 – 70.

[2] Bulitko, V. and Voorhees, B. (2008) Index permutations and classes of additive CA rules with isomorphic STD. Journal of Cellular Automata (to appear)

[3] Bulitko, V. and Voorhees, B. (2008) Cycle structure of baker transformation in one dimension. Submitted to Journal of Cellular Automata.

Elementary coupled cellular automata with memory

Ramón Alonso-Sanz[1,2] and Larry Bull[1]

[1] Department of Computer Science, University of the West of England, Bristol
Frenchay Campus. Bristol BS16 1QY, UK
[2] ETSI Agrónomos (Estadística), C.Universitaria. 28040, Madrid, Spain

{Ramon.Alonso-Sanz, Larry.Bull}@uwe.ac.uk

Abstract. In standard Cellular Automata (CA) the new state of a cell depends upon the neighborhood configuration only at the preceding time step. The effect of implementing memory capabilities in cells of coupled elementary CA with homologous cells interacting is studied in this article.

1 Cellular automata with memory

Cellular Automata (CA) are discrete, spatially explicit extended dynamic systems. A CA system is composed of adjacent cells or sites arranged as a regular lattice, which evolves in discrete time steps. Each cell is characterized by an internal state whose value belongs to a finite set. The updating of these states is made simultaneously according to a common local transition rule involving only the neighborhood of each cell [14, 21]. Thus, if $\sigma_i^{(T)}$ is taken to denote the value of cell i at time step T, the site values evolve by iteration of the mapping: $\sigma_i^{(T+1)} = \phi\big(\{\sigma_j^{(T)}\} \in \mathcal{N}_i\big)$, where $\mathcal{N}_i$ is the set of cells in the neighborhood of i and ϕ is an arbitrary function which specifies the cellular automaton *rule* operating on $\mathcal{N}_i$.

This article deals with two possible state values at each site: $\sigma \in \{0, 1\}$. As an example, the left spatio-temporal pattern in Fig. 1 shows the evolution of the conventional parity rule with three inputs (nearest neighbors arranged in a one-dimensional lattice plus self-interaction): $\sigma_i^{(T+1)} = \sigma_{i-1}^{(T)} \oplus \sigma_i^{(T)} \oplus \sigma_{i+1}^{(T)}$, starting from a single live cell. Despite its formal simplicity, the parity rule may exhibit complex behaviour [16].

In standard CA the transition function depends on the neighborhood configuration of the cells only at the preceding time step. Historic memory can be embedded in the CA dynamics by featuring every cell by a mapping of its states in the previous time steps. Thus, what is here proposed is to maintain the transition rules ϕ unaltered, but make them act on the cells featured by a function of their previous states: $\sigma_i^{(T+1)} = \phi\big(\{s_j^{(T)}\} \in \mathcal{N}_j\big)$, $s_j^{(T)}$ being a state function of the series of states of the cell j up to time-step T.

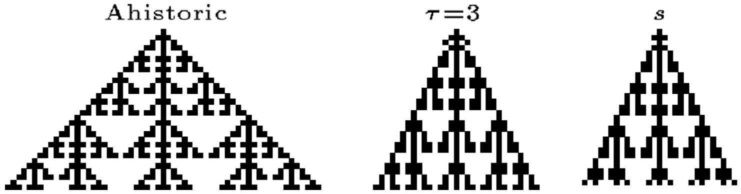


Fig. 1. The ahistoric parity rule (left) and this rule with memory of the majority of the last three states (center).

The central pattern in Fig. 1 shows the effect of endowing cells with memory of the most frequent of their three last states (reference to the last three states will be often shorted as τ=3) will be often on the parity rule with three neighbors. Thus, in the automaton with memory in Fig. 1, $\sigma_i^{(T+1)} = s_{i-1}^{(T)} \oplus s_i^{(T)} \oplus s_{i+1}^{(T)}$, it is : $s_i^{(T)} = mode\left(\sigma_i^{(T)},\sigma_i^{(T-1)},\sigma_i^{(\mathrm{T}-2)}\right)$. Memory alters the ahistoric evolution in Fig. 1 already at $T = 4$, because at $T = 3$, the actual pattern : ■ ■ ■ , differs from that of the featured cells (patterns headed s) : ■ , in which the outer live cells in the actual pattern (once alive, twice dead) are featured as dead. So the pattern at $T = 4$ is with memory : ▬, instead of the wider ahistoric pattern : ▬ ■ ▬ . As a result, memory truncates the expansive evolution of the parity rule.

This kind of majority memory exerts a characteristic inertial effect. The effect of memory on CA has been studied in previous work [2, 1].

2 Coupled Cellular Automata

Many systems of interest may be viewed as consisting of multiple networks, each with their own internal structure and *coupling* structure to their external world partners. Examples include economic markets, social networks, ecologies, and within organisms such as neuron-glia networks.

An easy way of coupling two networks of the same size is that of connecting their homologous cells. This is so in the one-dimensional networks of the subsequent figures.

Thus, the parity rule in Fig. 2 remains acting on cells with three inputs : their nearest neighbors in their own layer and their homologous cell in the partner layer. Noting σ and $[\sigma]$ the current state values and s and $[s]$ the trait states in left and right layers respectively, in the formulation of the parity rule in Fig. 2 it is : $\sigma_i^{(T+1)} = \sigma_{i-1}^{(T)} \oplus [\sigma_i^{(T)}] \oplus \sigma_{i+1}^{(T)}$, $[\sigma_i^{(T+1)}] = [\sigma_{i-1}^{(T)}] \oplus \sigma_i^{(T)} \oplus [\sigma_{i+1}^{(T)}]$, whereas with memory : $\sigma_i^{(T+1)} = s_{i-1}^{(T)} \oplus [s_i^{(T)}] \oplus s_{i+1}^{(T)}$, $[\sigma_i^{(T+1)}] = [s_{i-1}^{(T)}] \oplus s_i^{(T)} \oplus [s_{i+1}^{(T)}]$, with $s_i^{(T)} = mode\left(\sigma_i^{(T)},\sigma_i^{(T-1)},\sigma_i^{(\mathrm{T}-2)}\right)$, $[s_i^{(T)}] = mode\left([\sigma_i^{(T)}],[\sigma_i^{(T-1)}],[\sigma_i^{(\mathrm{T}-2)}]\right)$, and initial assignations $s_i^{(1)} = \sigma_i^{(1)}, s_i^{(2)} = \sigma_i^{(2)}$, $[s_i^{(1)}] = [\sigma_i^{(1)}], [s_i^{(2)}] = [\sigma_i^{(2)}]$.

Initially only the central cell of the left layer is active in Fig. 2 , in which evolution is shown up to T=30 . The restraining effect of the short-range memory

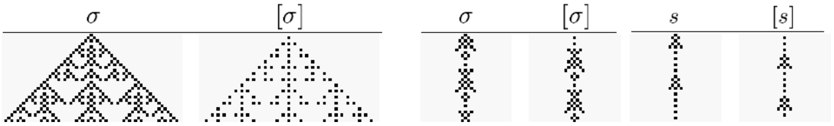


Fig. 2. The ahistoric coupled parity rule (left) and this rule with memory of the majority of the last three states (center). The trait states are shown in the right spatio-temporal patterns.

of the majority of the last three states becomes apparent also in the context of coupled automata.

Stochastically coupled CA and Boolean Networks have been implemented to study synchronization of coupled chaotic systems [13, 15, 17, 18] . The reduction induced by memory in the perturbation from a single live cell means a reduction in their differences of the spatio-temporal patterns of coupled CA. That might be an indicator of a higher synchrony in the dynamics of coupled CA with memory in general. But this is not necessarily so. At least not in the case of the *vivid* parity rule, as indicated in Fig. 3 by the differences in the patterns of coupled CA starting from the same random initial configurations and periodic boundary conditions imposed on the border cells.

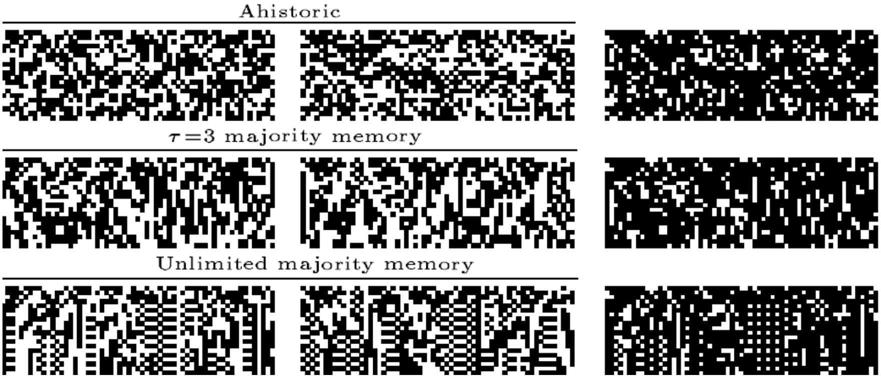


Fig. 3. Coupled parity CA starting at random. Ahistoric model, and models with τ=3 and unlimited majority memories. The last column of graphics shows the differences in the patterns of the two coupled layers.

Cells with unlimited majority memory in Fig. 3 are featured as,

$$s_i^{(T)} = mode\left(\sigma_i^{(T)}, \sigma_i^{(T-1)}, \ldots, \sigma_i^{(1)}\right),$$

$$[s_i^{(T)}] = mode\left([\sigma_i^{(T)}], [\sigma_i^{(T-1)}], \ldots, [\sigma_i^{(1)}]\right).$$

As a rule, increasing the length of the majority memory produces an increasing inertial effect, so that in the unlimited trailing majority memory scenario, oscillators or quasi-oscillators, tend to be generated, most of them of short length. This effect can be envisaged in Fig. 3, but becomes apparent starting for a single seed, as shown in the patterns under α=1 in Fig. 24 which actually corresponds to this scenario.

A preliminary study of the effect of memory on coupled CA, mainly of radio two, was made in [4].

In can be noted that an extension of the Java based CA system *JCAsim* [10] supports coupled CA simulation [7] .

3 Perturbations

Two coupled CA with identical transition rules, wiring scheme and initial state configuration will, of course, evolve identically. In the case of the inter-wiring scheme being that considered here, i.e., connection of homologous cells, the evolution of two K=2+1 coupled networks is that of a K=3 one-layer CA with self-interaction of the kind in Fig. 1.

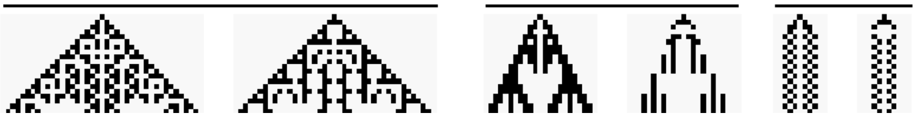


Fig. 4. Coupled parity rule with the central cell of the left network without connection with its homologous cell. Ahistoric (left), τ=3 (center) and unlimited trailing memory (right) models. Both central cells are initially at state 1.

In Fig. 4 both networks are initiated with an active central cell. But the ahistoric and τ=3 memory patterns in Fig. 4 are not those of Fig. 1 because there is a default in the coupling: the central cell of the left network does not receive the input from its homologous cell in the right one. Because of that it is not active at T=2 in the left layer. This simple default in wiring notably alters the patterns compared to that in Fig. 1.

Figures 5 and 6 show the evolving patterns in coupled twin layers of size 122 up to T=20, from the same random initial configuration. The common spatio-temporal pattern is shown in Figs. 5 and 6 with the active cells in grey tone. Darker cells show the *damage* (rather perturbation [20]) induced by two different causes : *i*) the just described lack in the inter-connection of the central cell of left network in Fig. 5, and *ii*) the reversion of the state value of this cell. Qualitatively speaking, the restraining effect of memory on the control of both types of perturbation spreading is similar. In fact much of the characteristics observed starting from a single seed may be extrapolated to the general damage spreading scenario: starting from a single site is a limit case of damage in fully quiescent automata.

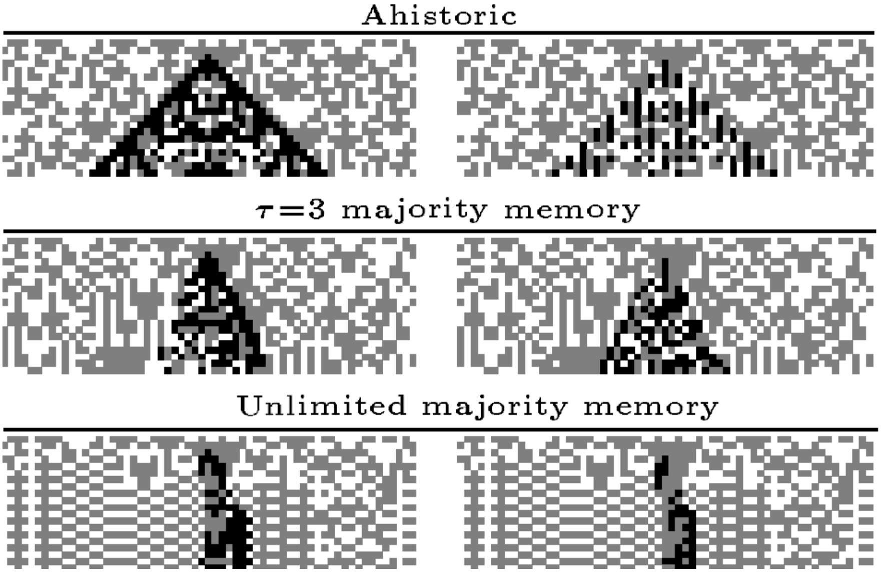


Fig. 5. Effect of the lack of inter-connection of the central cell of the left layer. Parity rule acting on twin layers.

4 Coupled automata with elementary rules

Elementary rules operate on nearest neighbors: $\sigma_i^{(T+1)} = \phi_i\big(\sigma_{i-1}^{(T)}, \sigma_i^{(T)}, \sigma_{i+1}^{(T)}\big)$. These rules are characterized by a sequence of binary values (β) associated with each of the eight possible triplets $\big(\sigma_{i-1}^{(T)}, \sigma_i^{(T)}, \sigma_{i+1}^{(T)}\big)$:

$$\begin{array}{c} 111\ \&;\ 110\ \&;\ 101\ \&;\ 100\ \&;\ 011\ \&;\ 010\ \&;\ 001\ \&;\ 000\ \&; \\ \beta_1\ \&;\ \beta_2\ \&;\ \beta_3\ \&;\ \beta_4\ \&;\ \beta_5\ \&;\ \beta_6\ \&;\ \beta_7\ \&;\ \beta_8 x \end{array} .$$

The *rule number* of elementary CA , $\mathcal{R} = \sum_{i=1}^{8} \beta_i 2^{8-i}$, ranges from 0 to 255. *Legal* rules are *reflection symmetric* (so that 100 and 001 as well as 110 and 011 yield identical values), and *quiescent* $(\beta_8 = 0)$. These restrictions leave 32 possible legal rules of the form: $\beta_1\beta_2\beta_3\beta_4\beta_2\beta_6\beta_4 0$.

The far right column of patterns in Fig. 7 shows the effect of τ=3 majority memory in the legal rules significantly affected by memory when starting form a single site seed. The overall effect of memory is that of restraining the expansion

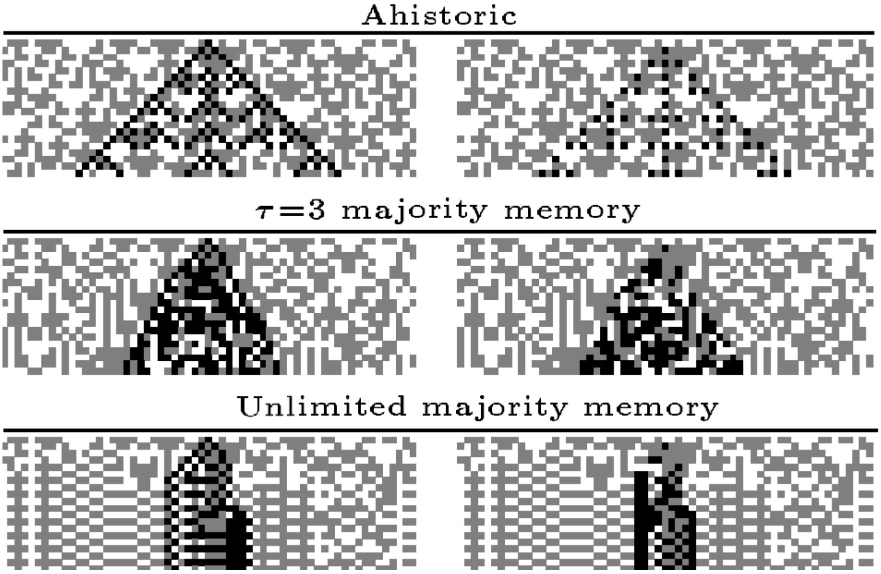


Fig. 6. Effect of reversing the state value of the central cell of the left layer in the scenario of Fig. 5.

of the spatio-temporal patterns, with particular incidence in rules 18, 90, 146, and 218 which extinguish as soon as $T{=}4$.

Coupled elementary rules evolve as:

$$\sigma_i^{(T+1)} = \phi\big(\,\sigma_{i-1}^{(T)}, [\sigma_i^{(T)}], \sigma_{i+1}^{(T)}\big) \quad , \quad [\sigma_i^{(T+1)}] = \phi\big(\,[\sigma_{i-1}^{(T)}], \sigma_i^{(T)}, [\sigma_{i+1}^{(T)}]\big) \ .$$

Automata with transition rules with no self-interaction in their one-layer formulation, i.e., $\sigma_i^{(T+1)} = \phi\big(\,\sigma_{i-1}^{(T)}, \sigma_{i+1}^{(T)}\big)$ [3], are unaltered if coupled in the simple way considered here. Examples are the left and right shift rules 170 and 240 $\big(\sigma_i^{(T+1)}{=}\sigma_{i+1}^{(T)}\,,\ \sigma_i^{(T+1)}{=}\sigma_{i-1}^{(T)}\big)$ and rule 90: $\sigma_i^{(T+1)} = \sigma_{i-1}^{(T)} \oplus \sigma_{i+1}^{(T)}$, which evolves in Fig. 7 *isolated* as in the one-layer scenario.

Rules with $\beta_6{=}\beta_8{=}0$, i.e., those which do not generate active cells when the neighbors are not active (regardless of the state of the cell itself) in the one-layer scenario, do not *transmit* activity to an initially fully inactive layer when coupled. Among the 64 elementary rules with $\beta_6{=}\beta_8{=}0$, are the legal rules 18,90,146,218, 50,122,178,250 reported in Fig. 7, and the *majority* rule 232. Also are neighbor-quiescent the asymmetric shift rules 170 and 240 (equivalent under the reflection transformation), and rules 226 and 184 (equivalent under the negative transformation).

Figure 7 shows under the heading -coupled- the evolving patterns of coupled legal elementary rules significantly affected by memory when starting from a

[3] Some authors refer to these rules as rules without *memory*.

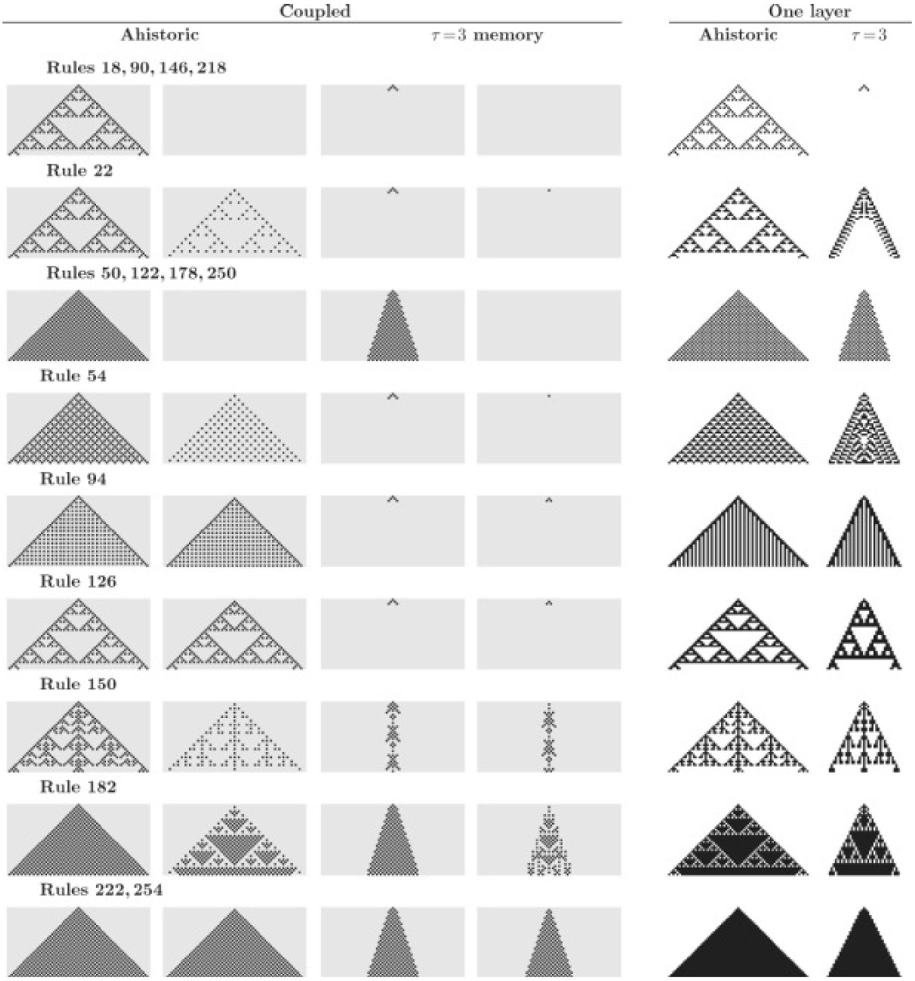


Fig. 7. Coupled and one-layer elementary legal CA starting from a single active cell in the left lattice. Ahistoric and mode of the last three state memory models.

single seed in one layer (the left one in Fig. 7). In this context, memory kills as in the ahistoric model, the evolution of the group of rules 18, 90, 146, and 218, and also that of rules 22, 54, 94, and 126. Memory notably narrows the evolution of the parity rule 150 (as advanced in Fig. 2), and also very much alters that of rules 182, 222, and 254.

The ahistoric patterns of rules 22, 54, 94, 126, 150, 182, 222, and 254 become split in two complementary ones in their coupled evolution in Fig. 7. This splitting effect in the ahistoric model affects to every quiescent rule when starting from a single active cell, so that the mere superposition (OR operation without the need of the XOR) of the patterns in the coupled formulation generates the ahistoric ones.

Let us point here that the split of the one-layer pattern starting from a single cell when coupling is not a particular property of elementary rules, but of the simple way of coupling adopted here. Thus for example it holds with the parity rule with five inputs as shown in Fig. 8 .

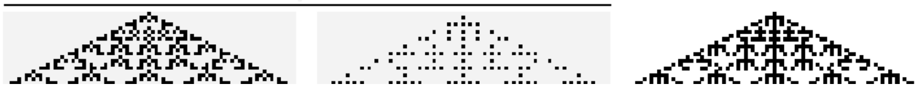


Fig. 8. The parity rule with five inputs. Coupled (overlined left) and one-layer (right) ahistoric models .

Starting at random from a sole layer, only the linear rules 60 (00111100) , $\sigma_i^{(T+1)} = \sigma_i^{(T)} \oplus \sigma_{i-1}^{(T)}$, and its reflected rule 102 (01100110), $\sigma_i^{(T+1)} = \sigma_i^{(T)} \oplus \sigma_{i+1}^{(T)}$, together with rules 150 and 204 , retain this superposition property in the ahistoric context. In Fig. 9 only the left layer starts with live cells, thus $\sigma_i^{(2)}{=}\sigma_{i-1}^{(1)} \oplus \sigma_{i+1}^{(1)}$, and $[\sigma_i^{(2)}]{=}\sigma_i^{(1)}$, so that the initial configuration is transmitted to the right layer after the first iteration. Then, at $T{=}3$, $[\sigma_i^{(3)}]{=}\sigma_{i-1}^{(1)} \oplus (\sigma_{i-1}^{(1)} \oplus \sigma_{i+1}^{(1)}) \oplus \sigma_{i+1}^{(1)} = 0$, and consequently, $[\sigma_i^{(4)}]{=}\sigma_i^{(3)}$. The left layer is at $T{=}3$, $\sigma_i^{(3)}{=}(\sigma_{i-2}^{(1)} \oplus \sigma_i^{(1)}) \oplus \sigma_i^{(1)} \oplus \sigma_i^{(1)} \oplus \sigma_{i+2}^{(1)})$, ${=}\sigma_{i-2}^{(1)} \oplus \sigma_i^{(1)} \oplus \sigma_i^{(1)} \oplus \sigma_{i+2}^{(1)}$, i.e., the same state value as in the one-layer configuration. The ulterior dynamic generalizes these initial results, so that the initially empty layer is empty at odd time-steps and copies that of the partner layer at even time-steps, whereas the configuration of the initially active layer at odd time-steps is the same as in the one-layer model. Qualitatively, the same dynamic is followed in rules 60 and 102, as seen in Fig. 9 .

Unlike in Fig. 7, starting at random as in Fig. 9 to get the one-layer patterns from those in the coupled formulation addition is necessarily to be done modulo two with the linear rules 60, 102 and 150 .

With $\tau{=}3$ majority memory, no rule, except the identity rule 204, $\sigma_i^{(T+1)} = \sigma_i^{(T)}$, verify this superposition property. Thus, as an example, the patterns of the coupled partner layers in Fig. 10 do not add the one-layer ones shown also in the figure. The identity rule 204 is unaffected by memory, so that in the coupled formulation the initial pattern appears at odd time steps in the left layer and at the even ones in the right layer, generating by direct superposition the one-layer lattice as shown in Fig. 9 .

Linear rules are *additive*, i.e., any initial pattern can be decomposed into the superposition of patterns from a single site seed. Each of these configurations can be evolved independently and the results superposed (module two) to obtain the final complete pattern. Mathematically, the distributive property holds for

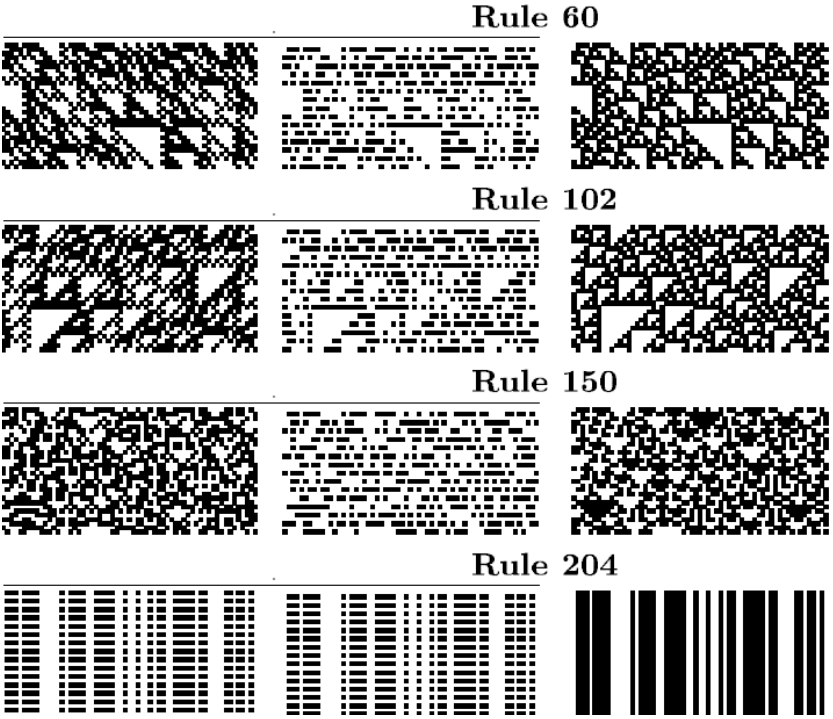


Fig. 9. Coupled elementary legal CA starting at random in the left lattice (overlined left) which generate via superposition the one-layer evolution (right).

every pair of initial configurations **u** and **v** : $\phi(\mathbf{u}\oplus\mathbf{v}) = \phi(\mathbf{u})\oplus\phi(\mathbf{v})$. The upper group of spatio-temporal patterns in Fig. 11 shows an example.

Linear rules remain linear when cells are endowed with linear memory rules. Thus, endowing the parity rule of the three last states as memory in cells upon the coupled elementary linear rule 150 yields:

$$\sigma_i^{(T+1)} = s_{i-1}^{(T)} \oplus [s_i^{(T)}] \oplus s_{i+1}^{(T)} \quad , \quad [\sigma_i^{(T+1)}] = [s_{i-1}^{(T)}] \oplus s_i^{(T)} \oplus [s_{i+1}^{(T)}] \ ,$$

with, $s_{i+1}^{(T)} = \sigma_{i+1}^{(T)} \oplus \sigma_{i+1}^{(T-1)} \oplus \sigma_{i+1}^{(T-2)}$, $s_i^{(T)} = \sigma_i^{(T)} \oplus \sigma_i^{(T-1)} \oplus \sigma_i^{(T-2)}$, $s_{i-1}^{(T)} = \sigma_{i-1}^{(T)} \oplus \sigma_{i-1}^{(T-1)} \oplus \sigma_{i-1}^{(T-2)}$, and, $[s_{i+1}^{(T)}] = [\sigma_{i+1}^{(T)}] \oplus [\sigma_{i+1}^{(T-1)}] \oplus [\sigma_{i+1}^{(T-2)}]$, $[s_i^{(T)}] = [\sigma_i^{(T)}] \oplus [\sigma_i^{(T-1)}] \oplus [\sigma_i^{(T-2)}]$, $[s_{i-1}^{(T)}] = [\sigma_{i-1}^{(T)}] \oplus [\sigma_{i-1}^{(T-1)}] \oplus [\sigma_{i-1}^{(T-2)}]$.

Consequently, linear rules remain additive when endowed with linear memory rules. Thus in the lower group of spatio-temporal patterns in Fig. 11 , the evolution patterns starting from the left layer (upper row), and from the right one (central row), if added modulo two generate the evolving pattern starting from both layers active (bottom row).

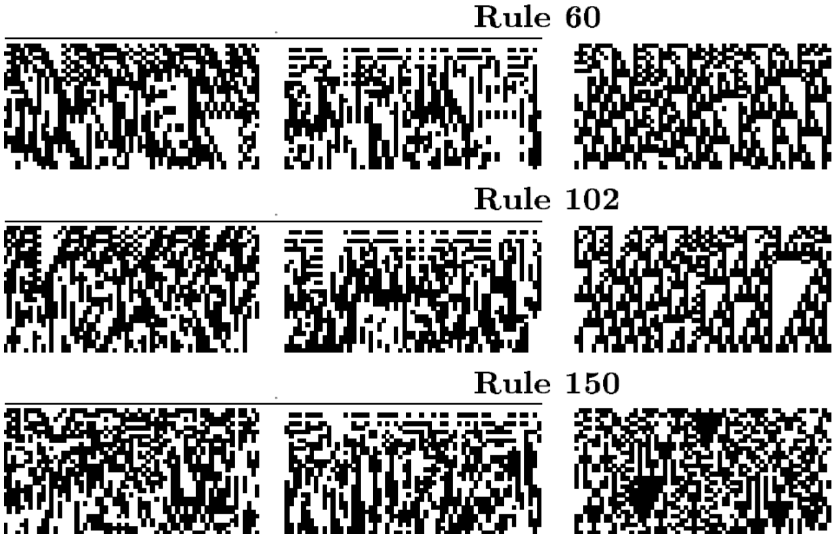


Fig. 10. Coupled elementary CA rules starting at random in the left lattice (overlined left), and one-layer (right) evolution with τ=3 majority memory.

5 Number conserving rules

The only non-trivial elementary rules conserving the number of active sites are the *traffic* rule 184 (10111000) and its reflected rule 226 (11001000) [6, 8]. These rules when coupled to an empty layer evolve as a left and right shift respectively[4], thus still conserving the total number of live cells. Starting from different random configurations in both layers, both rules 184 and 226 lose their characteristic number conserving property, in respect to the individual layers, but not considering the two layers as a whole, albeit presenting very different spatio-temporal patterns, as shown in Fig. 12 for rule 226 in a lattice of size 400 up to T=100.

When implementing majority memory, rules 184 and 226 lose the number conserving property, so that the pattern drifts to a fixed point of all 1s or all 0s depending upon whether within the configuration the number of 1s was superior to that of 0s or the contrary. The variation in the number of live cells in Fig. 13 is low (less than ten cells) because the initial density is fairly close to the watershed 0.5, so that only by T=1000, the pattern full blackens. The cases of low and high initial densities in Fig. 14 show that the rule 226 with τ=3 majority memory readily relaxes in both cases to a fixed point of all 0s or all 1s, correctly classifying

[4] $\begin{array}{cccc}101 & 100 & 001 & 000\\ 1 & 1 & 0 & 0\end{array}$, equivalent to $\sigma_i^{(T+1)}=\sigma_{i-1}^{(T)}$, and $\begin{array}{cccc}101 & 100 & 001 & 000\\ 1 & 0 & 1 & 0\end{array}$, equivalent to $\sigma_i^{(T+1)}=\sigma_{i+1}^{(T)}$.

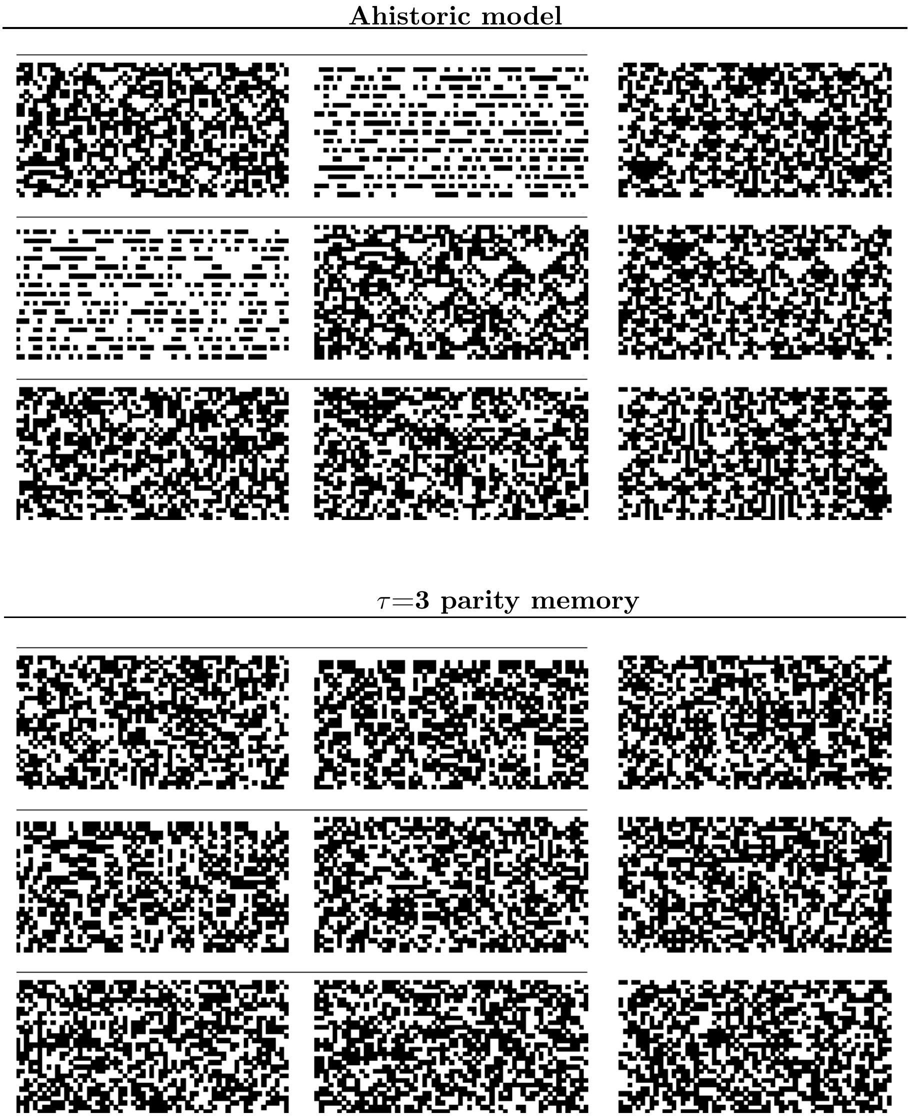


Fig. 11. Additivity of the linear rule 150 in the coupled (overlined left) and in the one-layer (right) scenarios. Ahistoric (upper group of patterns) and τ=3 parity (lower group of patterns) memories.

the initial configuration. This is not so when coupling the two halves of the one-layer scenario due to the presence of permanent solitons of the alternative state value. The relaxation of the criterion by which the rules 184 and 226 recognizes majority density [8] must bounce back when these rules are coupled.

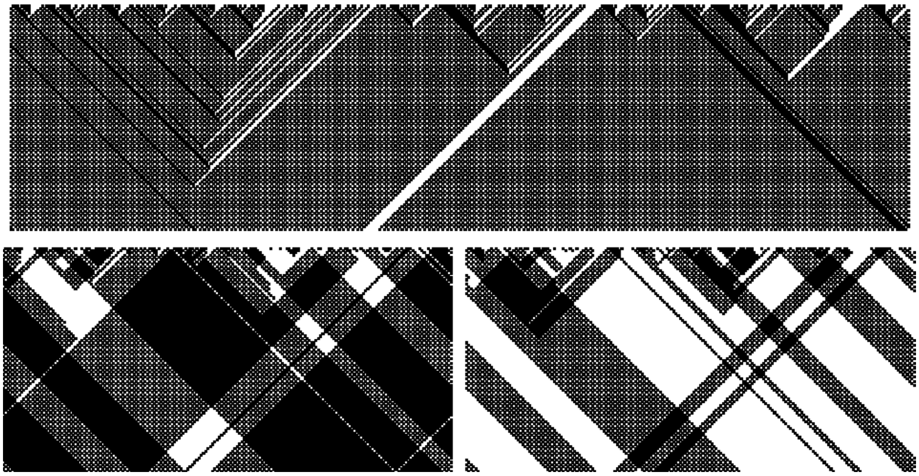


Fig. 12. Elementary rule 226 starting at random. One layer (upper) and coupled half (lower) layers.

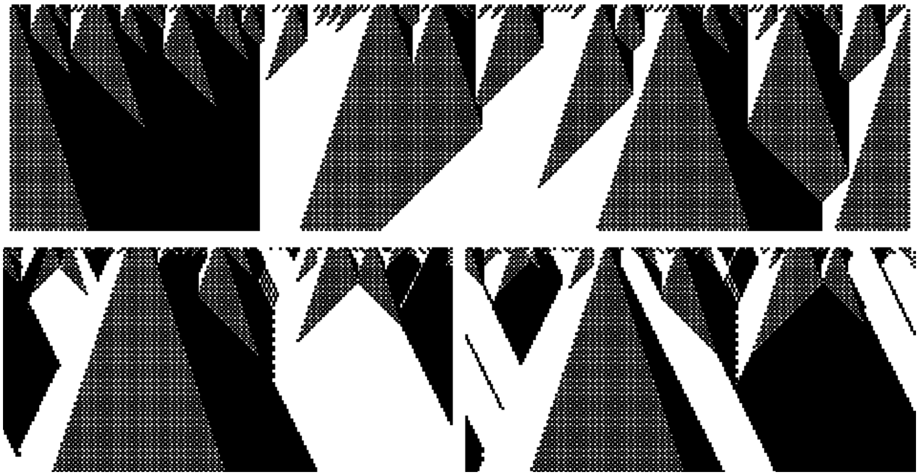


Fig. 13. Elementary rule 226 in the scenario of Fig. 12 with τ=3 mode memory.

The *majority* rule 232 together with the equivalent rules 184 and 226 has proved to be particularly effective among elementary rules in dealing with the *density* problem [9, 19] : to decide whether an arbitrary initial configuration contains a density of 1*s* above or below a given value σ_c, particularly when σ_c=0.5 . As with synchronization, the density task comprises a non-trivial computation for small-radius CA : density is a global property of a configuration, where small-radius CA rely solely on local interactions. starting to investigate this issue with the help of evolutionary based searching tools.

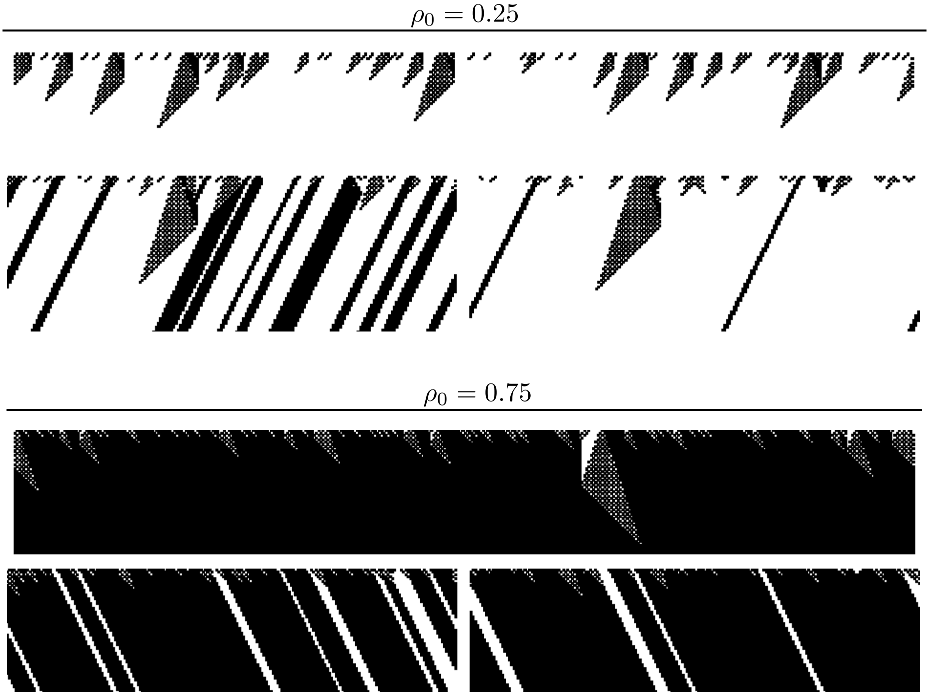


Fig. 14. The scenario of Fig. 13 from initial densities 0.25 and 0.75 .

So far, we expect that rules endowed with memory in cells in the form of the mode of the three last states acting in the temporal domain will produce good results in classifying density, as suggested here with respect to the elementary rules 184 and 226 . In the case of rule 226 by means of the examples of the evolution of the spatio-temporal patterns in Fig. 13 and in the case of rule 184 by the dynamics of the density shown in Fig. 15. It becomes apparent from Fig. 15 that rule 184 with τ=3 very soon relaxes to the correct fixed point if the initial density is either $\rho \geq 0.6$ or $\rho \leq 0.4$. In the $[0.4, 0.6]$ interval the drift of density is much slower, but tending to the steady state that marks *correct* classification. This is so even for the particular cases that appear stabilized up to T=300, as shown also in Fig. 15. Only one of the plots that finally evolve upwards in the lower graph of Fig. 15 corresponds to an initial density under 0.5, albeit very close to this limit value. Increasing the length of the trailing memory up to T=5 does not imply increasing the performance in the solution of the classification task. Beyond this trailing length, inertial effects opposite the drift to the all 1s or all 0s intended in the density classification. It seems that τ=3 as length of the majority memory is a good (and simple) choice regarding the density task.

Non-uniform or *hybrid* CA have been shown to be a good tool to solve the density task [19]. The CA with memory here may be also considered hybrid

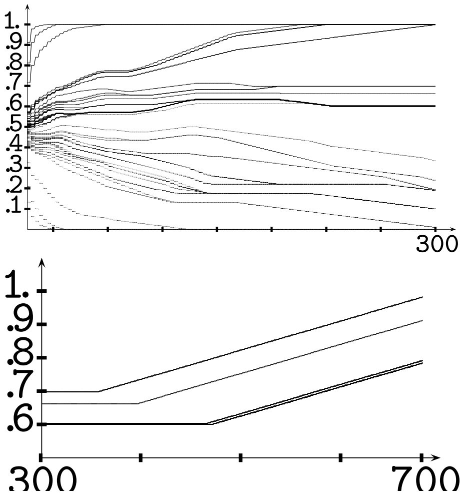


Fig. 15. Evolution of density under rule 184 up to T=300. Further evolution of the instances stable up to T=300 are shown below.

(or composite), but in space and time, implementing in the time context the most successful density task solver rule: the majority rule. It is expected that a synergetic effect will emerge, so that rules in the standard context that poorly deal with the density task problem, when combined with the majority memory become good density solvers.

The evolution of rule 262 with memory in Fig. 13 (and that of rule 184 shown in [4]) resembles that of the r=3 Gacs-Kurdyumov-Levin (GKL [11, 12]) rule [5], whose evolution in the one-layer and coupled models is shown in Fig. 16 starting from the same initial configuration as in Fig. 12.

Both GKL and rules with τ=3 mode memory have in common the presence of the mode operation and produce excellent results in the density task.

Eventually number-conserving (ENC) CA reach limit sets, after a number of time steps of order of the automaton size, having a constant number of active sizes [5]. Some ENC rules *emulate* proper number-conserving rules, albeit this is not a necessary condition. Among the ENC which emulate NC stand the number-nonincreasing rule 176, which emulates both rules 184 and 240, and the not monotone rule 99 which emulates rule 226, shown in Fig. 19 . When coupled, these rules also evolve as ENC rules, reaching a plateau after a fairly short transition period (see Figs. 17 and 18) . The numbers reached in the steady

[5]
$$\sigma_i^{(T+1)} = \begin{cases} mode\big(\sigma_i^{(T)}, \sigma_{i-1}^{(T)}, \sigma_{i-3}^{(T)}\big) \; amp; \; if \;\; amp; \sigma_i^{(T)} = 0 \\ mode\big(\sigma_i^{(T)}, \sigma_{i+1}^{(T)}, \sigma_{i+3}^{(T)}\big) \; amp; \; if \;\; amp; \sigma_i^{(T)} = 1 \end{cases}$$

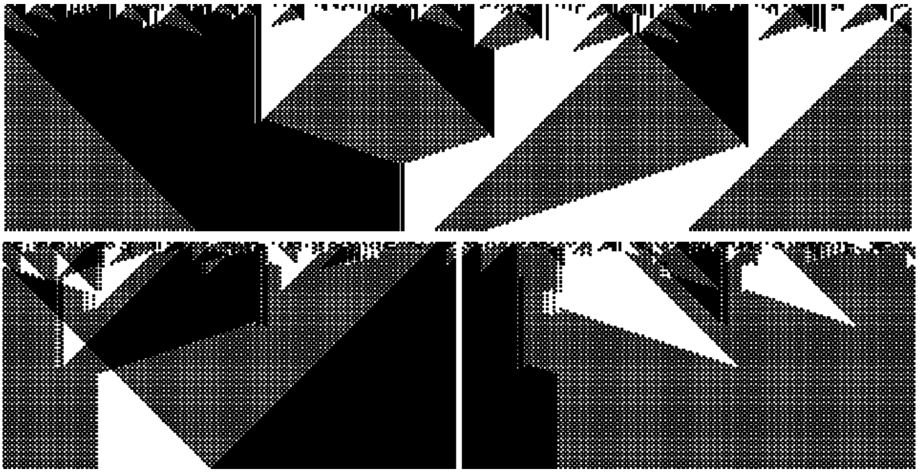


Fig. 16. The GKL rule starting at random. One layer (upper) and coupled half (lower) layers.

regime in other layer and coupled contexts are not exactly the same, albeit they use to be fairly close.

None of these rules with memory seem to solve the density task as rules 184 and 226 do, because the implementation of memory seems not to discriminate among the different levels of initial density. Thus, the evolution of density is altered in a similar way regardless of ρ_o, either depleting it as happens with rule 176 in Fig. 17, or *ii*) leaving it fairly unaltered as in the case of rule 99 in Fig. 18[6]. This is so regardless of the dramatic change that may suffer the spatio-temporal patterns, as shown in Fig. 19. The dynamics under rule 99 has been also checked endowing cells with memory of the parity of their three last states. Even in this scenario rule 99 shows a notable resilience to abandon the tendency to stabilize the density close to 0.5. The effect of memory on ENC rules that do not emulate NC rules, e.g. the not monotone rules 74 and 88 is similar to that just described.

6 Asymmetric memory

Figure 20 corresponds to an asymmetric scenario in which respect memory as only one of the layers (the right one in Fig. 20) has cells endowed with memory, whereas the cells of its partner layer are featured by their last state. Both coupled layers have its central cell active initially.

In both scenarios, short-range (τ=3) and unlimited majority memory, memory soon kills the dynamic in the layer with memory in the group of rules

[6] Under rule 99, the density departing from the highest values plummets initially in such a way that its change is hidden in the y-axis. Conversely, the variation from the smallest initial densities is hidden in the y-axis due to its initial dramatic increase.

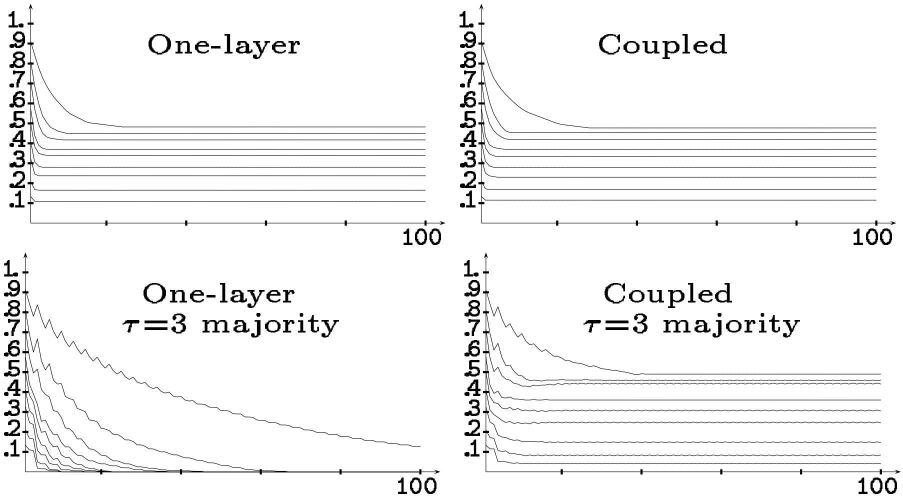


Fig. 17. The evolution of density under rule 176 .

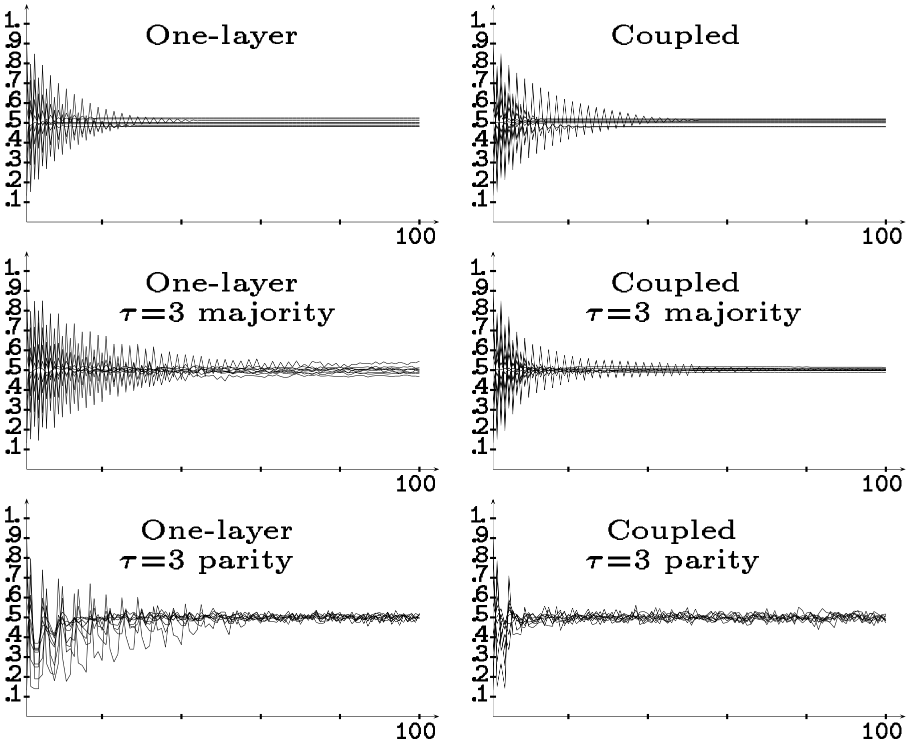


Fig. 18. The evolution of density under rule 99 .

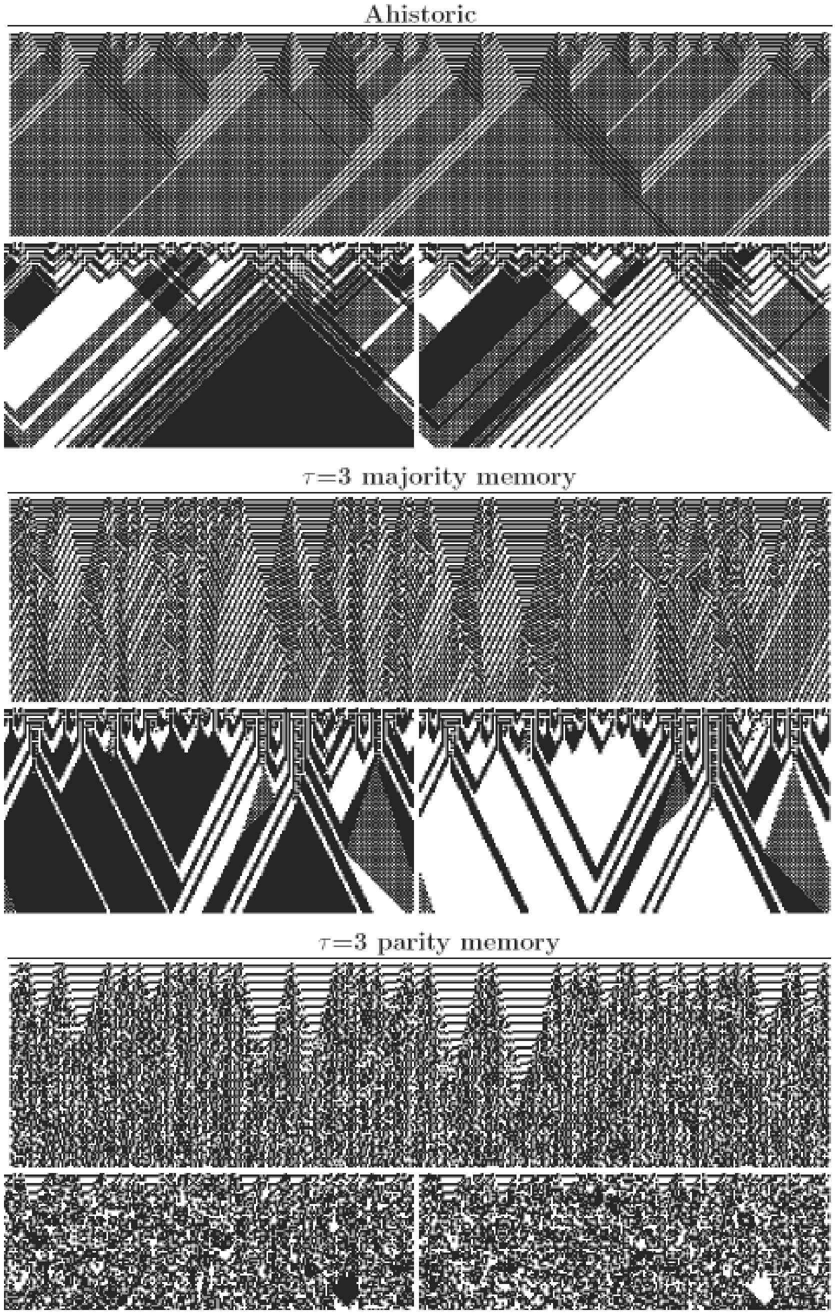


Fig. 19. The rule 99. One layer (upper) and coupled half (lower) layers.

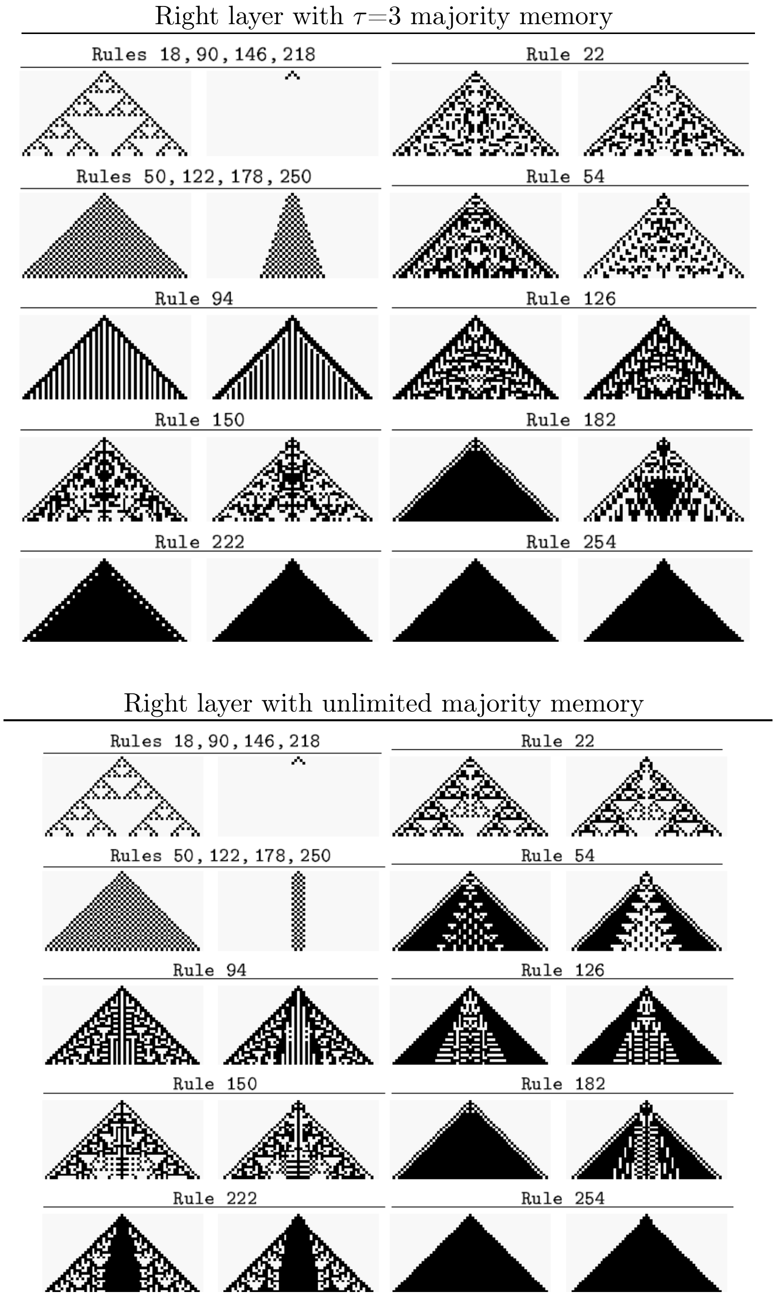


Fig. 20. Coupled legal rules with only the right layer endowed with memory.

18,90,146,218, and excerpts the characteristic restraining effect in the case of rules 50,122,178, and 250. But the remaining rules, rule 150 in particular, do not evolve in a restrained form, so that the ahistoric networks seem to drive the process. This evolution is rather unexpected, particularly in the case of unlimited majority memory, a memory implementation very much tending to *freeze* any dynamical system.

7 Elementary rules as memory

Any Boolean function of previous state values, other than parity or majority, may act as memory rule. Elementary rules in particular, so that: $s_i^{(T)} = \phi\big(\sigma_i^{(T_1)}, \sigma_i^{(T)}, \sigma_i^{(\mathrm{T}-2)}\big)$, $[s_i^{(T)}] = \phi\big([\sigma_i^{(T_1)}], [\sigma_i^{(T)}], [\sigma_i^{(\mathrm{T}-2)}]\big)$.

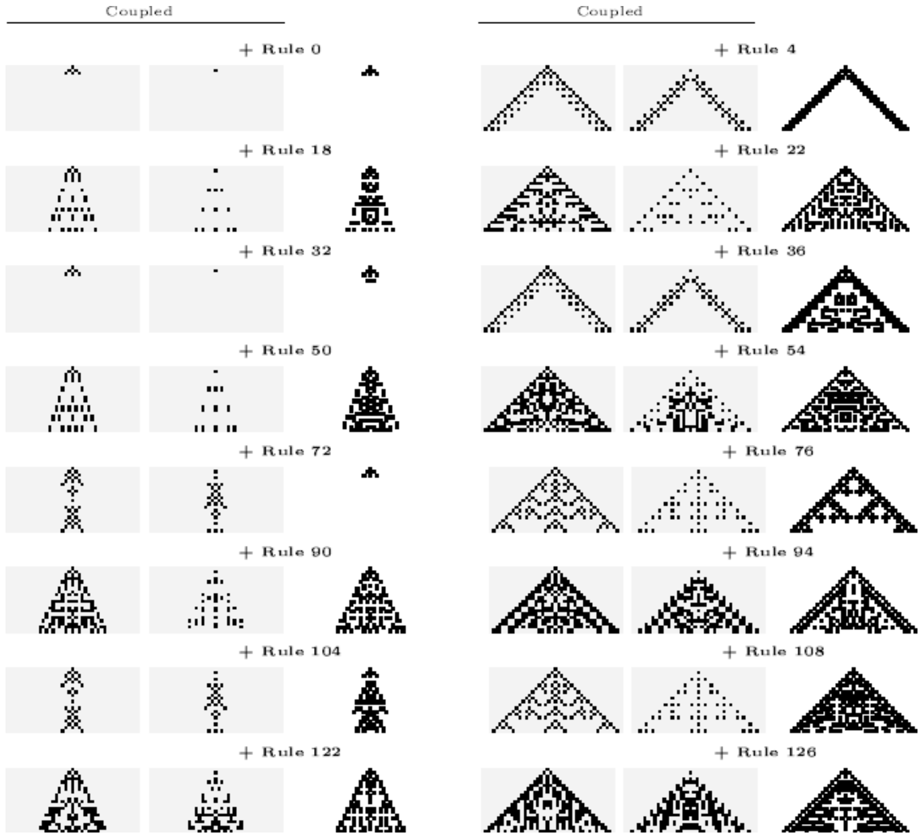


Fig. 21. Effect legal rules as memory on rule 150 .

Figure 21 shows the effect of legal rules as memory on rule 150 , and Fig. 22 reports the effect of quiescent asymmetric rules of low number as memory also

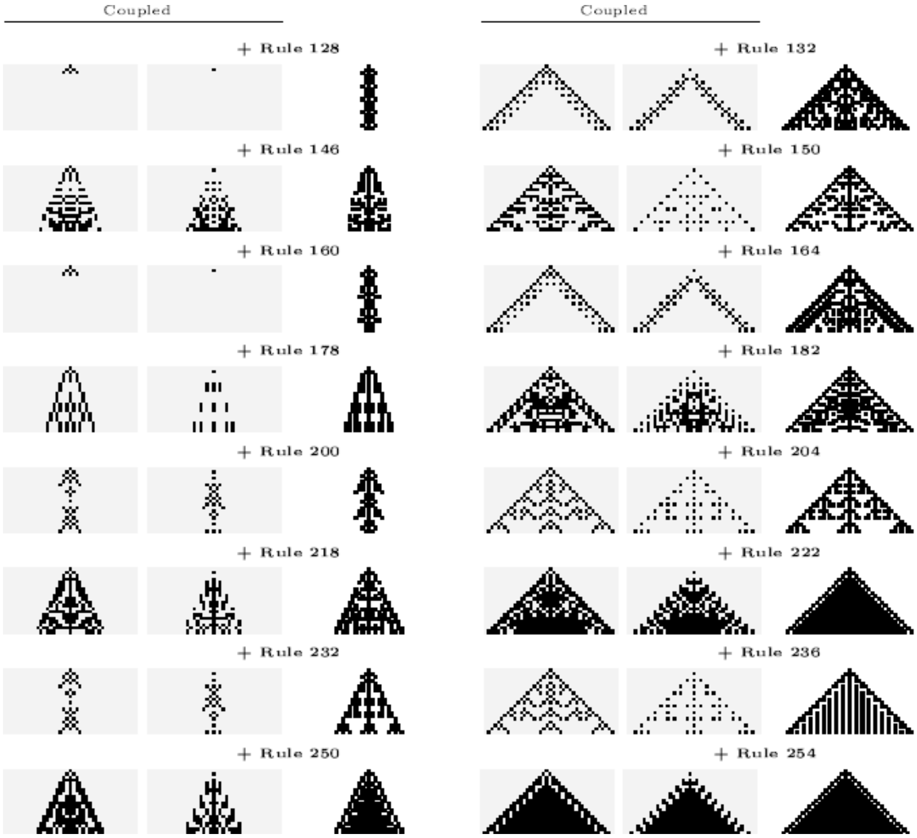


Fig. 21. (*Continued*).

on rule 150. Initially $s_i^{(1)} = \sigma_i^{(1)}, s_i^{(2)} = \sigma_i^{(2)}$, $[s_i^{(1)}] = [\sigma_i^{(1)}], [s_i^{(2)}] = [\sigma_i^{(2)}]$ in these figures. In the fairly endless patchwork of patterns generated as a result of the composition of the spatial rule 150 with the elementary rules acting of memory, the *original* aspect of the rule 150 is not always traceable.

8 Average memories

The historic memory of every cell i can be weighted by applying a geometric discounting process based on the *memory factor* α lying in the [0,1] interval, so that:

$$m_i^{(T)}(\sigma_i^{(1)}, \ldots, \sigma_i^{(T)}) = \frac{\sigma_i^{(T)} + \sum\limits_{t=1}^{T-1} \alpha^{T-t} \sigma_i^{(t)}}{1 + \sum\limits_{t=1}^{T-1} \alpha^{T-t}} \equiv \frac{\omega_i^{(T)}}{\Omega(T)} \quad .$$

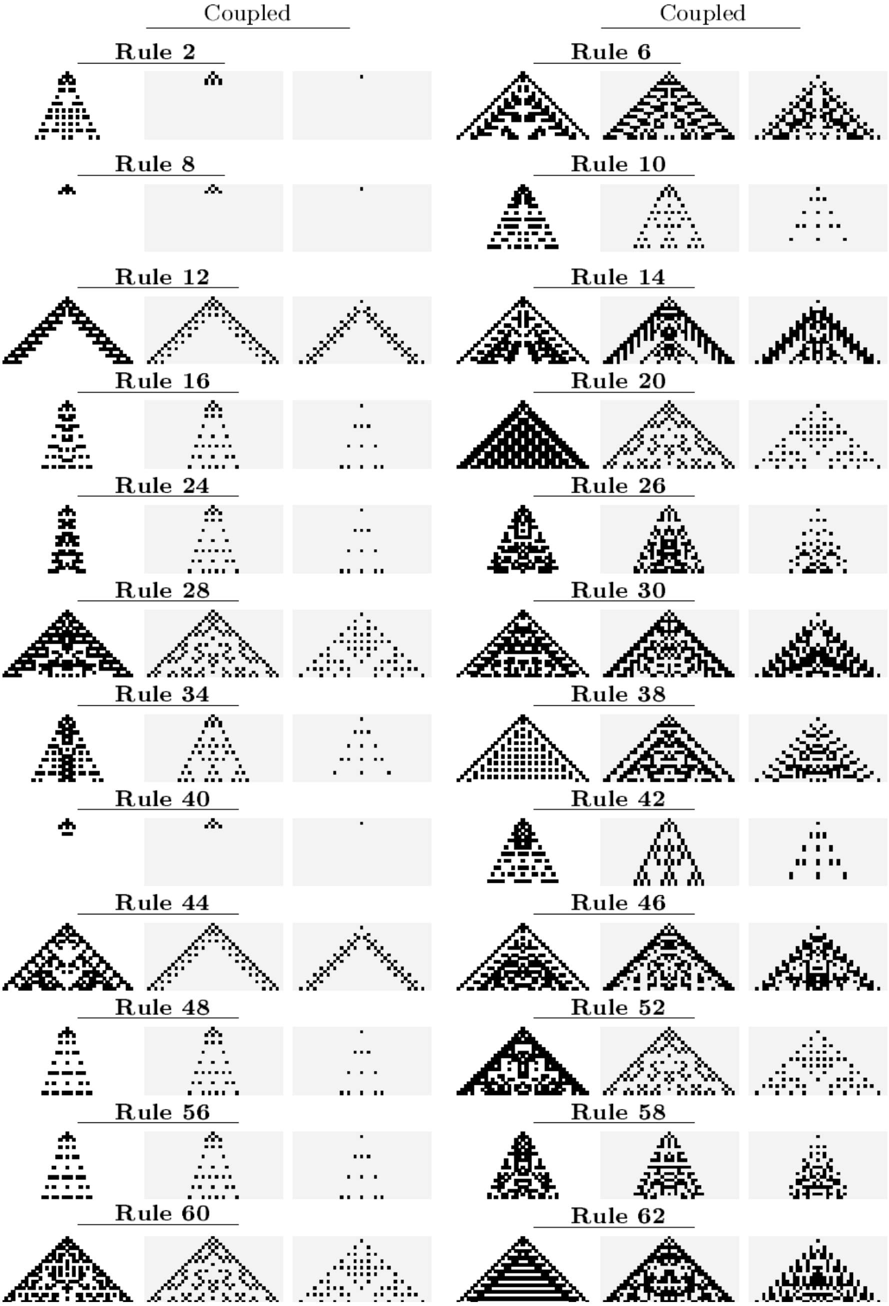


Fig. 22. Effect of quiescent asymmetric rules as memory on rule 150 .

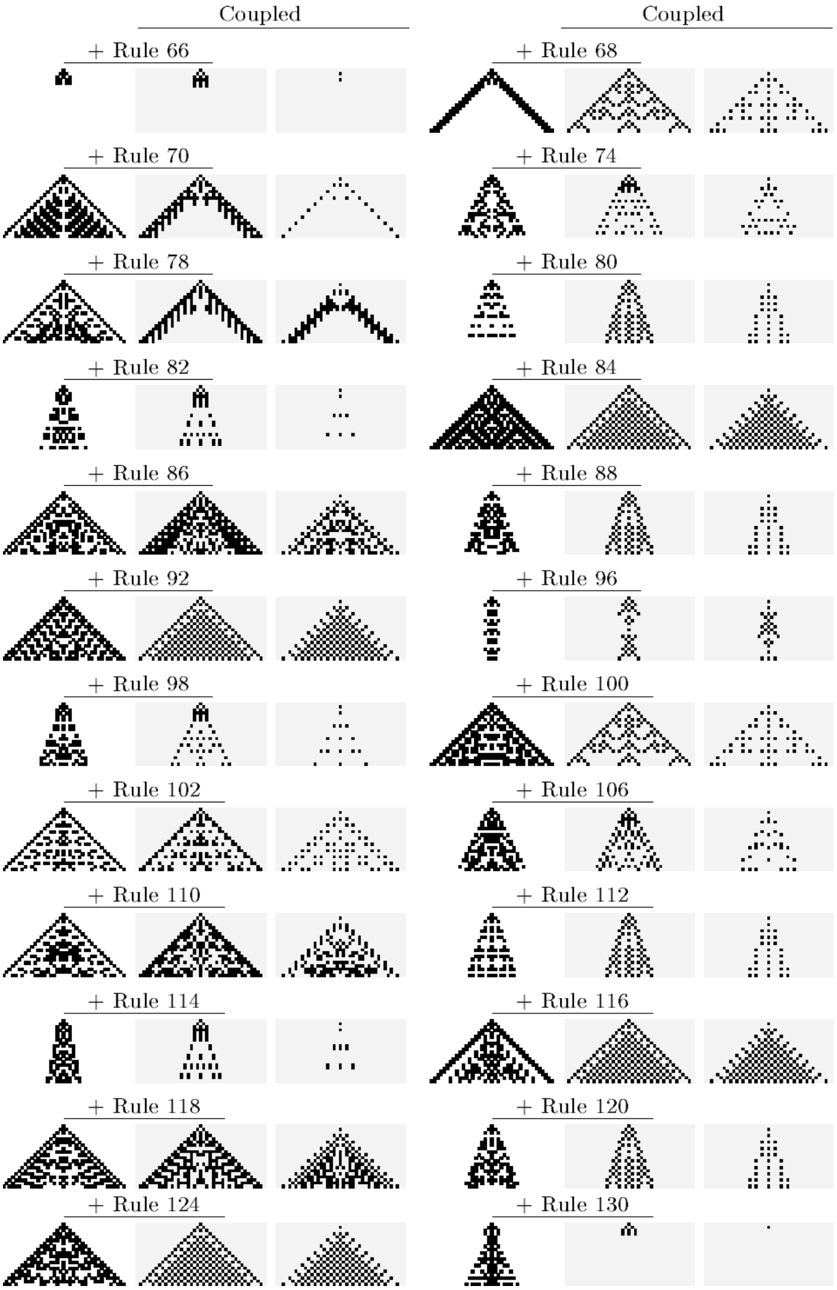


Fig. 22. (*Continued*).

This well known mechanism fully takes into account the last round, and tends to *forget* the older rounds. The trait state (s) will be obtained by comparing the not rounded weighted mean memory *charge* m to the landmark 0.5 (if $\sigma \in \{0,1\}$), assigning the last state in case of an equality to this value. Thus:

$$s_i^{(T)} = \mathcal{H}(m_i^{(T)}) = \begin{cases} 1 & \text{if } m_i^{(T)} > 0.5 \\ \sigma_i^{(T)} & \text{if } m_i^{(T)} = 0.5 \\ 0 & \text{if } m_i^{(T)} < 0.5. \end{cases}$$

The choice of the memory factor α simulates the long-term or remnant memory effect: the limit case $\alpha = 1$ corresponds to a memory with equally weighted records (*full* memory, equivalent to the *mode* if $k = 2$), whereas $\alpha \ll 1$ intensifies the contribution of the most recent states and diminishes the contribution of the more remote states (short-term working memory). The choice $\alpha = 0$ leads to the ahistoric model.

In the most unbalanced scenario, $\sigma_i^{(1)} = ... = \sigma_i^{(T-1)} \neq \sigma_i^{(T)}$, it holds that: $m = 0.5 \Rightarrow \alpha_T^T - 2\alpha_T + 1 = 0$ [7]. At $T = 3$, it is $\alpha_3^3 - 2\alpha_3 + 1 = 0 \Rightarrow \alpha_3 = 0.61805$, and at $T = 4$, it is $\alpha_4 = 0.5437$. When $T \to \infty$, the effectivity equation becomes: $-2\alpha_\infty + 1 = 0$, thus, due to the rounding operation $\mathcal{H}$, in the $k = 2$ scenario, α-memory is not effective if $\alpha \leq 0.5$.

Figure 23 shows the restraining effect of increasing values of the memory factor α on rule 150 from a single site seed up to T=30. As stated, α-memory is not effective at T=3, altering the pattern at T=3, unless $\alpha \geq \alpha_3 = 0.61805$, so in Fig. 23 for the values $\alpha \geq 0.7$. Memory is effective at T=4 in the interval $[0.55, 0.6]$, i.e., with $\alpha \geq \alpha_4 = 0.5437$.

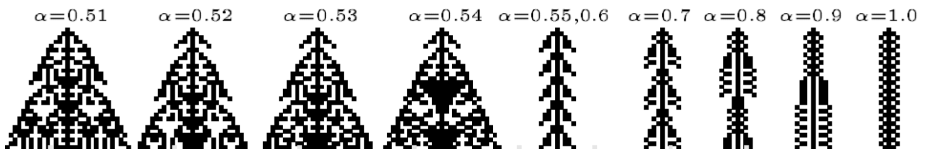


Fig. 23. Effect of α-memory on rule 150 from a single cell.

Figure 24 shows the effect of increasing α-memory in the coupled automata scenario. The lower values of α approach the τ=3 majority memory effect, but higher values very much confine the perturbation. Frequently, full memory (α=1.0) tends to generate oscillators as in Figure 24.

[7]
$$\left\{ \begin{array}{l} m(0,0,\ldots,0,1) = \dfrac{1}{\dfrac{\alpha^T - 1}{\alpha - 1}} = \dfrac{1}{2} \\[2ex] m(1,1,\ldots,1,0) = \dfrac{\dfrac{\alpha^T - \alpha}{\alpha - 1}}{\dfrac{\alpha^T - 1}{\alpha - 1}} = \dfrac{1}{2} \end{array} \right\} \Rightarrow \alpha_T^T - 2\alpha_T + 1 = 0$$

It is remarkable that this memory mechanism is not *holistic* but *cumulative* in its demand for knowledge of past history: the whole $\{\sigma_i^{(t)}\}$ series needs not be known to calculate the term $\omega_i^{(T)}$ of the memory *charge* $m_i^{(T)}$, while to (sequentially) calculate $\omega_i^{(T)}$ one can resort to the already calculated $\omega_i^{(T-1)}$ and compute: $\omega_i^{(T)} = \alpha\omega_i^{(T-1)} + \sigma_i^{(T)}$.

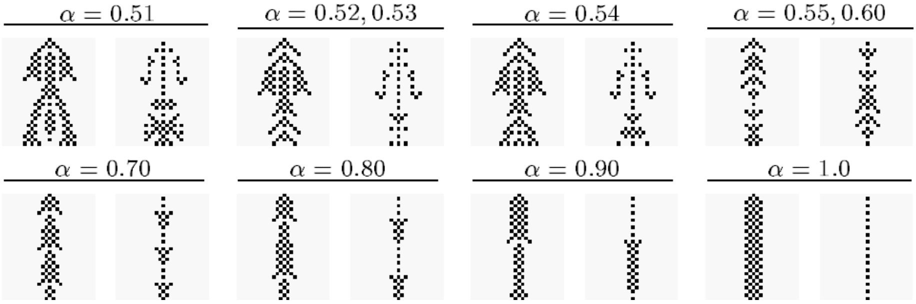


Fig. 24. Effect of α-memory in the scenario of Fig. 2 .

Another memory mechanism, similar to that used in the context of connection weights adjustments in neural networks, is one in which the distance between the state value and the actual one is adjusted with the so called *learning rate* δ. In the Widrow-Hoff like scheme implemented here, the trait state s is obtained by rounding the average m incremented by the pondered discrepancy between the *expected* average value m and the current state σ: $s_i^{(T)} = \mathcal{H}\big(m_i^{(T)} + \delta(\sigma_i^{(T)} - m_i^{(T)})\big)$. Thus, δ=1 leads to the ahistoric model, whereas δ=0 leads to the memory model based on the average m Figure 25 shows the effect of such kind of memory in the scenario of Fig. 1 , with m being the unweighted average: $m_i^{(T)} = \dfrac{1}{T}\sum_{t=1}^{T}\sigma_i^{(t)}$.

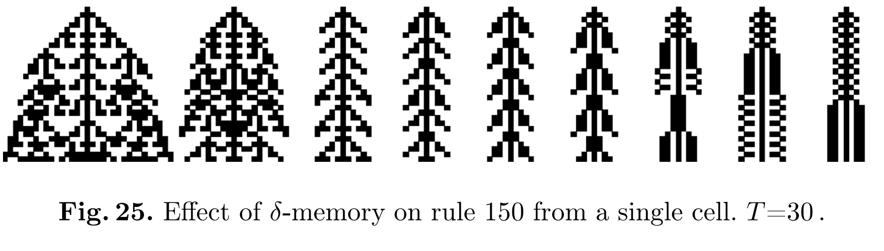


Fig. 25. Effect of δ-memory on rule 150 from a single cell. T=30 .

In the most unbalanced scenario, in the δ-memory mechanism it holds that: $m = 0.5 \Rightarrow \delta_T = \frac{1}{2}\frac{T-2}{T-1}$ [8]. It is, δ_3=1/4, thus in Fig. 25 the pattern at T=4 differs from the ahistoric one (as in Fig. 1) only for $\delta \leq$0.2 and 0.1 . It is δ_4=1/3 , thus the pattern at T=5 in Fig. 25 is the ahistoric one when δ=0.35 but is altered under δ=0.30 . When $T \to \infty$, it is: $\delta_\infty = \frac{1}{2}$, thus δ-memory is not effective if $\delta \geq 0.5$.

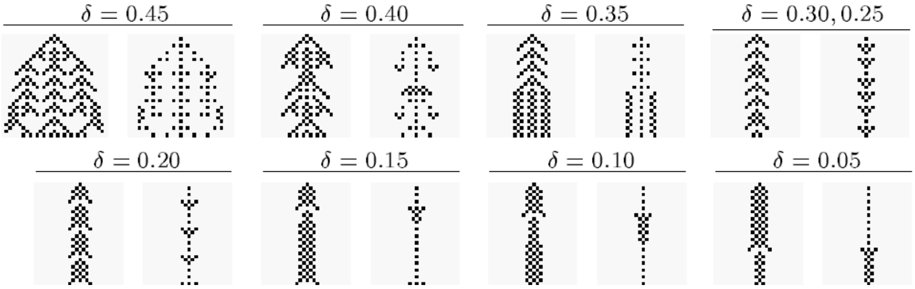


Fig. 26. Effect of δ-memory in the scenario of Fig. 2 .

Figure 26 shows the effect of decreasing values of δ-memory in the coupled CA context of Fig. 2 . Evolution under the higher δ values approaches to that of τ=3 majority memory, whereas δ=0.1 approaches that of the unlimited full trailing memory.

9 Conclusion

Embedding memory in cells of coupled cellular automata notably alters the dynamics compared to the conventional ahistoric one. As a general rule, memory of average kind, such as majority memory, tends to inhibit *complexity*, inhibition that can be modulated by varying the depth of memory, but non-average type memory, such as parity memory, opens a notable new perspective in CA.

A major impediment in modeling with CA stems from the difficulty of utilizing their complex behavior to exhibit a particular behavior or perform a particular function. CA with memory in cells broadens the spectrum of CA as a tool for modeling, in a fairly natural way for easy computer implementation. It is likely that in some contexts, a transition rule with memory could match the

[8]
$$\left.\begin{cases} m(0,0,\ldots,0,1) = \frac{1}{T} + \delta(1-\frac{1}{T}) = \frac{1}{2} \\ m(1,1,\ldots,1,0) = \frac{T-1}{T} + \delta(0-\frac{T-1}{T}) = \frac{1}{2} \end{cases}\right\} \Rightarrow \delta_T = \frac{1}{2}\frac{T-2}{T-1}$$

correct behavior of the CA system of a given complex system. This could mean a potential advantage for CA with memory (CAM) over standard CA as a tool for modeling. Anyway, besides their potential applications, CAM have an aesthetic and mathematical interest on their own. A further modification as explored here, is to couple CAM thereby opening the possibility of other new dynamics which may be suitable for modeling various systems.

The study of the effect of memory on coupled disordered CA and Boolean Networks [3] is planned as a forthcoming work.

References

[1] Alonso-Sanz, R. Reversible CA with memory, *Physica D*, 175,(2003),1. One-dimensional, r=2 CA with memory, *Int.J. Bifurcation and Chaos*, 14,(2004),3217. Phase transitions in an elementary probabilistic CA with memory, *Physica A*, 347,(2005),383.

[2] Alonso-Sanz, R. and Martin, M. Elementary CA with memory, *Complex Systems*, 14,(2003),99. Three-state one-dimensional CA with memory, *Chaos, Solitons and Fractals*, 21,(2004),809. One-dimensional CA with memory in cells of the most frequent recent value, *Complex Systems*, 15,(2005),203. Elementary CA with elementary memory rules in cells,*J. of Cellular Automata*,(2006), 1,70.

[3] Alonso-Sanz, R. and Cárdenas, J. P.(2007). On the effect of memory in disordered Boolean networks: the K=4 case, *Int.J. Modern Physics C*, 18,1313. The effect of memory in cellular automata on scale-free networks: the $\overline{K}$=4 case, *Int.J. Bifurcation and Chaos.*, (2008),(in press).

[4] Alonso-Sanz, R. and Bull, L.(2008) One-dimensional coupled cellular automata with memory: initial investigations. *J. of Cellular Automata* (in press).

[5] Boccara, N.(2007) Eventually number-conserving cellular automata. *Int. J. Modern Physycs C,* ,18,35-42.

[6] Boccara, N., and Fuks, H.(2002) Number-conserving cellular automaton rules. *Fundamenta Informaticae* ,20,1-13 .

[7] Briesen, M. and Weimar, J.R.(2001) Distribute simulation environment for coupled cellular automata. In *PARCO 2001* . Joubert,G. *et al.* (eds). Imperial College Press.

[8] Fuks, H. and Sullivan, K.(2007) Enumeration of number-conserving cellular automata rules with two inputs. *Journal of Cellular Automata* 2,2,141-148.

[9] Fuks, H.(1997) Solution of the density classification problem with two cellular automata rules. *Phys. Rev. E* ,55,R2081-R2084.

[10] Freiwald, U. and Weimar, J.R.(2000) The Java based cellular automata simulation sistem - JCAsim. *Fut. Gen. Comp. Systems*, ,18,995-1004.

[11] Gacs, P.,Kurdyumov, G.L. and Levin, L. A.(1978) One-dimensional uniform arrays that wash out finite islands. *Problemy Pederachi Infomatsii* ,14,92-98 .

[12] Gonzaga de Sá, P. and Maes, C.(1992) The Gacs-Kurdyumov-Levin automaton revisited. *Journal of Statistical Physics* ,67,3/4,507-522 .

[13] Grassberger, P.(1999) Chaos and diffusion in deterministic cellular automata. *Phy. Rev. E* ,59,R2520 .

[14] Ilachinski, A. *CA. A Discrete Universe,* (World Scientific, 2000).

[15] Inoue, M. and Nagadome, M.(2000) Edge of chaos in stochastically coupled cellular automata. *J. Phys. Soc. Jpn.* , 69,1-4 .

[16] Julian, P. and Chua, L.O.(2002) Replication properties of parity cellular automata, *Int. J. Bifurcation and Chaos*, 12,477-94.

[17] Morelli, L.G. and Zanetti, D.H.(2001) Synchronization of Kaufmann networks. *Phsyical Review E*, 63, 036204-10 .

[18] Morelli, L.G. and Zanetti, D.H.(1998) Synchronization of stochastically coupled cellular automata. *Phsyical Review E*, 58,1, R8-R11 .

[19] Sipper, M. *Evolution of Parallel Cellular Machines* (Springer, 1997).

[20] Stauffer, D.(1994) Evolution by damage spreading in Kauffman model, *J. Stat. Physics*, 74, 5/6, 1293-1299.

[21] Wolfram, S. *A New Kind of Science.* (Wolfram Media, 2002).

On undecidability of sensitivity of reversible cellular automata*

Ville Lukkarila[1][2]

[1] Turku Centre for Computer Science
FI-20520 Turku
Finland
[2] Department of Mathematics
University of Turku
FI-20014 Turku
Finland
vinilu@utu.fi

Abstract. It has been shown earlier by Durand, Formenti and Varouchas that equicontinuity and sensitivity are undecidable properties for one-dimensional cellular automata. It has been shown by Kari and Ollinger that equicontinuity is undecidable for reversible cellular automata also. In this paper it is shown that sensitivity is undecidable for reversible cellular automata.

1 Introduction

Cellular automata are a simple formal model for the study of phenomena caused by local interaction of finite objects. A cellular automaton consists of a regular lattice of cells. Each cell has a state which is updated on every time step according to some local rule which is the same for all the cells in the lattice. The locally used update rule is simply called a local rule. On every time step the next state of the cell is determined according to the its own previous state and the previous states of a finite number of its neighbors. The state information of the entire lattice at any time step is called a configuration of the cellular automaton.

The cellular automata were introduced by von Neumann to study biologically motivated computation and self-replication [8]. The mathematical study of cellular automata in symbolic dynamics was initiated by Hedlund [3]. Although cellular automata may seem a simple model for computation, they can exhibit very complex behavior. A well-known example of such complex behavior is the Game-of-Life. Even though the rule according to which the lattice is updated is quite simple in the Game-of-Life, some state patterns interact in a somewhat complex manner. In fact, the Game-of-Life has been shown to be computationally universal. In particular, any Turing machine can be simulated with some cellular automaton in a natural way.

* This research has been supported by the Fund of Yrjö, Vilho and Kalle Väisälä.

Cellular automata have been studied very extensively also as discrete time dynamical systems. Injectivity, surjectivity, nilpotency, topological transitivity, topological mixing and different variants expansivity are among widely studied properties of cellular automata. In this paper some aspects of the equicontinuity classification (due to Kurka [6]) of one-dimensional cellular automata are reviewed. It is already known that equicontinuity and sensitivity are undecidable properties for irreversible cellular automata [2]. It has been recently shown by Kari and Ollinger that equicontinuity is undecidable even for reversible cellular automata [5]. It is known that regularity (i.e. regularity of the column subshifts as a formal language) is undecidable among irreversible cellular automata [7].

In this paper it is shown that sensitivity (i.e. lack of equicontinuity points) is undecidable for reversible cellular automata. Also, from the result of Kari and Ollinger it follows that regularity is undecidable even for reversible cellular automata.

2 Cellular automata

Cellular automata are dynamical systems which update the variables on an infinite d-dimensional lattice according to some function with a finite number of arguments. Formally, a *cellular automaton* is a 4-tuple $\mathcal{A} = (d, A, N, f)$, where d is the *dimension*, A is the *state set*, $N = (\overrightarrow{x_1}, \ldots, \overrightarrow{x_n})$ is the *neighborhood vector* consisting of vectors in $\mathbb{Z}^d$ and $f : A^n \to A$ is the *local rule*. A *configuration* $c \in A^{\mathbb{Z}^d}$ is a mapping which assigns a unique state for each cell location in $\mathbb{Z}^d$. The cells in locations $\overrightarrow{x} + \overrightarrow{x_i}$ are called *neighbors* of the cell in location $\overrightarrow{x}$.

At every time step the new configuration c' is determined by

$$c'(\overrightarrow{x}) = f(c(\overrightarrow{x} + \overrightarrow{x_1}), \ldots, c(\overrightarrow{x} + \overrightarrow{x_n})), \tag{1}$$

that is, the new state of cell in location $\overrightarrow{x}$ is computed by applying the local rule to its neighbors. The *global rule* $F : A^{\mathbb{Z}^d} \to A^{\mathbb{Z}^d}$ is defined by setting $F(c) = c'$ in the sense of equation 1.

A cellular automaton is said to be *reversible* if the global rule F has an inverse mapping F^{-1}. It can be shown that if the inverse mapping F^{-1} exists, it is a global rule of a cellular automaton, that is, it is defined by a local rule. It is also known that F is reversible if, and only if, it is injective. Furthermore, in the case of cellular automata injectivity of the global rule implies surjectivity of the global rule [4].

The distance between two different configurations c and e can be defined to be

$$d(c, e) = \left(\frac{1}{2}\right)^{\min\{\|\overrightarrow{x}\|_\infty \ \mid \ c(\overrightarrow{x}) \neq e(\overrightarrow{x})\}},$$

where $\| \cdot \|_\infty$ is the max-norm. Function $d(\cdot, \cdot)$ is also a metric thus making the set of configurations a metric space. The balls in the metric are called *cylinders* and they form a basis for the topology. Radius r cylinder centered at configuration

c is the set

$$\mathrm{Cyl}(c,r) = \left\{e \in A^{\mathbb{Z}^d} \mid c(\overrightarrow{x}) = e(\overrightarrow{x}) \text{ when } \|\overrightarrow{x}\|_\infty \leq r\right\}$$

For every radius r there are only finitely many cylinders and these cylinders are by definition disjoint. Therefore, radius r cylinders form a partition of the space of configurations. Hence, every cylinder is clopen because the complement of every cylinder is a union of other cylinders with the same radius. In the one-dimensional case, one can define cylinders as sets

$$\mathrm{Cyl}(w,k) = \left\{c \in A^{\mathbb{Z}} \mid c(i+k) = w(i) \text{ when } i \leq |w|-1\right\}$$

where $w \in A^*$ is a finite word and $w(i-1)$ denotes the ith letter of the word.

Pair (X,F) is a *dynamical system* if X is a compact metric space and $F : X \to X$ is a continuous mapping. In particular, d-dimensional cellular automata are dynamical systems of the form $(A^{\mathbb{Z}^d}, F)$.

A point $x \in X$ is an *equicontinuity point* of mapping F if for any $\varepsilon > 0$ there exists such $\delta > 0$ that for any point $y \in X$ and integer $n \in \mathbb{N}$,

$$d(x,y) < \delta \Longrightarrow d(F^n(x), F^n(y)) < \varepsilon.$$

A dynamical system (X,F) is *equicontinuous* if every point $x \in X$ is an equicontinuity point.

A word $w \in A^*$ is said to be *blocking* if there exists such a sequence of words $(w_n)_{n=0}^{\infty}$ that $F^n(\mathrm{Cyl}(w,i)) \subseteq \mathrm{Cyl}(w_n,0)$ for some integer i and any $n \in \mathbb{N}$ and word length $|w_n|$ is equal to the radius r of the local rule for any $n \in \mathbb{N}$. Set

$$\Sigma_k(F) = \left\{x \in \left(A^k\right)^{\mathbb{N}} \mid \exists y \in A^{\mathbb{Z}} : \forall i \in \mathbb{N} : F^i(y) \in \mathrm{Cyl}(x(i),0)\right\}$$

is called the *column subshift* of the one-dimensional cellular automaton $(A^{\mathbb{Z}}, F)$. A cellular automaton is *regular* if for any positive integer k the finite words (i.e. words over alphabet A^k) appearing in the sequences of $\Sigma_k(F)$ as factors form together a regular language. Equivalently, cellular automaton is regular if $\Sigma_k(F)$ is *sofic* for every k. It should be noted that sometimes also the denseness of periodic points is referred to as regularity [2]. However, here regularity means the property of the column subshifts according to Kurka [6].

A dynamical system (X,F) is *sensitive to initial conditions* (or *sensitive*) if there exists such $\varepsilon > 0$ that for any $x \in X$ and $\delta > 0$ there exists such a point $y \in X$ that

$$0 < d(x,y) < \delta \text{ and } d(F^n(x), F^n(y)) \geq \varepsilon$$

for some integer $n \in \mathbb{N}$. If the constant ε exists, it is known as the *sensitivity constant.* For one-dimensional cellular automata sensitivity is equivalent to the nonexistence of equicontinuity points.

A dynamical system (X,F) is *periodic* if there exists such an integer p that $F^{p+i} = F^i$ for every $i \in \mathbb{N}$. It is *ultimately periodic* if there exists such integers p_0 and p that $F^{p_0+p+i} = F^{p_0+i}$ for every $i \in \mathbb{N}$.

Theorem 2.1 ([1]). *A cellular automaton is equicontinuous if, and only if, it is ultimately periodic.*

Theorem 2.2 ([1]). *Any equicontinuity point has an occurrence of a blocking word. Conversely, any point with infinitely many occurrences of blocking words to the left and right of the origin is an equicontinuity point.*

3 Undecidability results

Theorem 3.1 ([5]). *It is undecidable whether a given one-dimensional reversible cellular automaton is equicontinuous, that is, all configurations are equicontinuity points.*

Using the method of Di Lena [7] with the undecidability of equicontinuity [5], one can show that regularity is an undecidable property even for reversible one-dimensional cellular automata.

Corollary 3.1. *It is undecidable whether a given one-dimensional reversible cellular automaton is regular.*

Proof (sketch). Assume that regularity is a decidable property among reversible cellular automata. A reversible cellular automaton is equicontinuous if, and only if, it is periodic. Every periodic cellular automaton is regular because the language of a column subshift consists of powers of a finite number of words and all their subwords.

Periodicity is a decidable property among regular cellular automata because the graph presentation can be constructed algorithmically for regular automata [7]. This contradicts the undecidability of periodicity. □

Theorem 3.2. *It is undecidable whether a given one-dimensional reversible cellular automaton is sensitive, that is, it has no equicontinuity points.*

Proof (sketch). Using the Turing machine simulation method given by Kari and Ollinger [5], a cellular automaton can be constructed which always has equicontinuity points. This cellular automaton can be further modified so that an additional set of states can pass through the already existing blocking word sequences if, and only if, the blocking word sequence does not contain a halting Turing machine simulation started on an empty tape. □

4 Acknowledgements

The financial support by the Fund of Yrjö, Vilho and Kalle Väisälä is gratefully acknowledged. The author is grateful to Professor J. Kari for guidance, useful discussions and advices.

References

[1] F. Blanchard and P. Tisseur. *Some properties of cellular automata with equicontinuity points.* Annales de l'Institute Henri Poincaré, 36(5):562–582, 2000.

[2] B. Durand, E. Formenti, and G. Varouchas. *On undecidability of equicontinuity classification for cellular automata.* Discrete Mathematics and Theoretical Computer Science, pages 117–128, 2003.

[3] G. Hedlund. *Endomorphisms and automorphisms of shift dynamical systems.* Math. Systems Theory, 3:320–375, 1969.

[4] J. Kari. *Theory of cellular automata: A survey.* Theoretical Computer Science, 334:3–33, 2005.

[5] J. Kari and N. Ollinger. Manuscript under preparation, 2008.

[6] P. Kurka. *Languages, equicontinuity and attractors in cellular automata.* Ergodic Theory and Dynamical Systems, 17:417–433, 2997.

[7] P. Di Lena. *Decidable propertied for regular cellular automata.* In Fourth IFIP conference on Theoretical Computer Science - TCS 2006, pages 185–196. Springer-Verlag, 2006.

[8] J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, 1966.

A 24-state universal one-dimensional reversible cellular automaton

Kenichi Morita

Hiroshima University, Graduate School of Engineering,
Higashi-Hiroshima, 739-8527, Japan

Abstract. We study the problem of finding a simple one-dimensional reversible cellular automaton (RCA) that have computation-universality. So far, a 30-state universal RCA that works on infinite (ultimately periodic) configurations has been given by Morita. Here, we improve this result, and show there is a 24-state universal RCA that also works on infinite configurations. Similar to the case of the 30-state RCA, the 24-state RCA simulates any cyclic tag system, which is a universal string rewriting system proposed by Cook.

1 Introduction

Since the early stage of the history of cellular automata (CAs), computation-universality has been studied as one of the important problems in CAs. As it is well known, von Neumann [13] designed a 29-state 2-D CA that is both computation- and construction-universal. Codd [3] improved the results, by giving a 8-state model. In these CAs, Turing machines can be realized as finite configurations. (Note that a “finite configuration" is a one such that the number of cells in non-quiescent state is finite.) Later, Banks [1] showed a 2-state Neumann-neighbor 2-D CA in which any Turing machine can be embedded as an ultimately periodic infinite configuration. On the other hand, it has bee shown that the Game of Life, a 2-state Moore-neighbor 2-D CA, is universal, where any 2-counter machine can be realized as a finite configuration [2].

In the case of 1-D CAs, Cook [4] proved that the elementary (i.e., 2-state 3-neighbor) CA of rule 110 with infinite configurations is computation-universal. As for 1-D universal CA with finite configurations, we can obtain a 7-state 3-neighbor model by slightly modifying the CA given by Lindgren and Nordahl [6]. An intrinsically universal 1-D CA, which can simulate any 1-D CA in infinite configurations, was studied by Ollinger and Richard [14], and they gave a 4-state 3-neighbor model.

As for “reversible" CAs (RCAs), Toffoli [15] first showed the existence of a computation-universal 2-D RCA. Later, Morita and Harao [8] showed that the class of 1-D RCAs is also computation-universal. In the case of 2-D RCA, various simple universal RCAs have been presented until now. Margolus [7] proposed an interesting 2-state 2-D RCA with Margolus-neighbor that can simulate the

Billiard-Ball Model of computation by which the Fredkin gate, a universal reversible logic gate, can be realized. Morita and Ueno [9] used the framework of partitioned CA (PCA), a subclass of conventional CAs, and gave two models of 16-state Neumann-neighbor universal 2-D reversible PCAs. Imai and Morita [5] showed a 8-state universal reversible triangular PCA that has an extremely simple local function. All the above universal 2-D RCAs need infinite configurations. Morita et al. [10] showed a 81-state universal 2-D reversible PCA in which any reversible 2-counter machine can be realized as a finite configuration.

Here, we investigate how simple 1-D universal RCAs can be. In [11] (also presented at AUTOMATA 2005) Morita proposed two such universal reversible PCAs. The first one is a 36-state model on infinite configurations, and the second one is a 98-state model on finite configurations. These models can simulate any cyclic tag system (CTAG), which is a kind of string rewriting system having universality proposed by Cook [4]. Later, the number of states of the former model was reduced, i.e., a 30-state universal reversible PCA was given in [12] (and in AUTOMATA 2006). In this paper, we further improve this result by showing a 24-state universal reversible PCA that works on infinite configurations.

2 Preliminaries

In this paper, we use the framework of partitioned cellular automata (PCAs) rather than that of conventional CAs, because the former makes it easy to design an RCA with a desired property.

Definition 2.1. *A deterministic one-dimensional three-neighbor* partitioned cellular automaton *(PCA) is defined by*

$$P = (\mathbf{Z}, (L, C, R), f),$$

where $\mathbf{Z}$ *is the set of all integers,* $L, C,$ *and* R *are non-empty finite sets of states of left, center, and right parts of each cell, and* $f : R \times C \times L \to L \times C \times R$ *is a* local function. *The next state of each cell is determined depending on the states of the right part of the left-neighboring cell, the center part of this cell, and the left part of the right-neighboring cell as shown in Fig. 1. Hereafter, the relation* $f(r, c, l) = (l', c', r')$ *is indicated as in Fig. 2, and called a* local rule *of* P*. The* global function $F : \mathbf{Z}^Q \to \mathbf{Z}^Q$ *(where* $Q = (L \times C \times R)$*) is defined by applying the local function* f *to all the cells of* P *in parallel (see e.g., [8, 11] for its detail). A PCA* P *is called* locally reversible *iff the local function* f *is injective, and called* globally reversible *iff the global function* F *is injective.*

In [8], it is shown that P is globally reversible iff locally reversible. Furthermore, it is easy to see that a PCA is a subclass of a conventional CA. Therefore, to obtain a reversible CA with some desired property, it is sufficient to give a locally reversible PCA with the property. Thus, the framework of a PCA facilitates to design an RCA. However, in general, the number of the states of a cell of a PCA becomes large, since the state set of each cell is $(L \times C \times R)$.

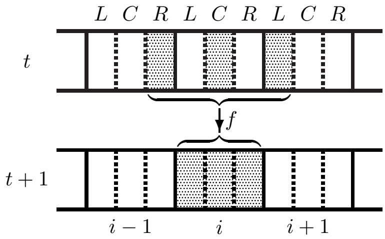


Fig. 1. The cellular space and a local function of a 1-D 3-neighbor PCA.

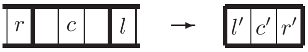


Fig. 2. A local rule of a 1-D 3-neighbor PCA.

Definition 2.2. *A* cyclic tag system *(CTAG) is a kind of string rewriting system having computation-universality [4]. It is defined by*

$$C = (k, \{Y, N\}, (p_0, \cdots, p_{k-1})),$$

where k ($k = 1, 2, \cdots$) is the length of a cycle (i.e., period), $\{Y, N\}$ is the (fixed) alphabet, and $(p_0, \cdots, p_{k-1}) \in (\{Y, N\}^*)^k$ is a k-tuple of production rules. A pair (v, m) is called an instantaneous description *(ID) of C, where $v \in \{Y, N\}^*$ and $m \in \{0, \cdots, k-1\}$. m is called a* phase *of the ID. A transition relation $\Rightarrow$ on the set of IDs is defined as follows. For any $(v, m), (v', m') \in \{Y, N\}^* \times \{0, \cdots, k-1\}$,*

$$\begin{array}{l} (Yv, m) \Rightarrow (v', m') \quad \textit{iff} \quad [m' = m + 1 \bmod k] \wedge [v' = vp_m], \\ (Nv, m) \Rightarrow (v', m') \quad \textit{iff} \quad [m' = m + 1 \bmod k] \wedge [v' = v]. \end{array}$$

A sequence of IDs $(v_0, m_0), (v_1, m_1), \cdots$ is called a computation starting from $v \in \{Y, N\}^*$ *iff $(v_0, m_0) = (v, 0)$ and $(v_i, m_i) \Rightarrow (v_{i+1}, m_{i+1})$ $(i = 0, 1, \cdots)$, and is denoted by $(v_0, m_0) \Rightarrow (v_1, m_1) \Rightarrow \cdots$.*

A CTAG is a system where production rules $(p_0, \cdots, p_{k-1})$ are used cyclically, i.e., p_m is used when the phase is m $(m = 0, \cdots, k-1)$. If the head symbol of a rewritten string is N, then the symbol N is simply removed. On the other hand, if the head symbol is Y, then it is removed and p_m is attached at the end of the rewritten string.

Example 2.1. The following is a simple example of a CTAG.

$$C_0 = (3, \{Y, N\}, (Y, NN, YN)).$$

If we give NYY to C_0 as an initial string, then

$$(NYY, 0) \Rightarrow (YY, 1) \Rightarrow (YNN, 2) \Rightarrow$$

$$\Rightarrow (NNYN, 0) \Rightarrow (NYN, 1) \Rightarrow (YN, 2) \Rightarrow \cdots$$

is a computation starting from NYY.

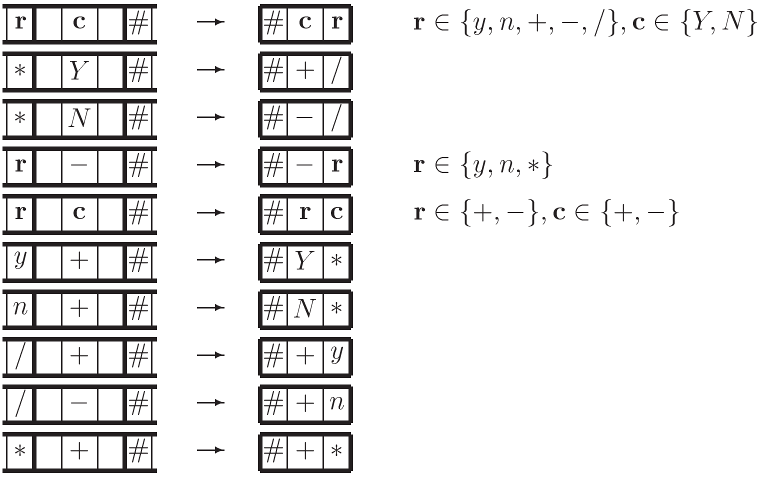
3 A 24-state universal reversible PCA P_{24}

The proposed model is a 24-state reversible PCA P_{24} that operates on ultimately periodic infinite configurations. It is defined as follows.

$$P_{24} = (\mathbf{Z}, (\{\#\}, \{Y, N, +, -\}, \{y, n, +, -, *, /\}), f_{24}).$$

The state set of each cell is $\{\#\} \times \{Y, N, +, -\} \times \{y, n, +, -, *, /\}$, and thus P_{24} has 24 states. Note that P_{24} is actually a two-neighbor (hence, one-way) PCA, since the state set of the left part is $\{\#\}$.

The local function f_{24} consists of the 24 local rules shown in Fig. 3. We can see there is no pair of distinct local rules whose right-hand sides are the same. Hence, f_{24} is injective, and thus P_{24} is an RCA.



$\mathbf{r}$	$\mathbf{c}$	$\#$	→	$\#$	$\mathbf{c}$	$\mathbf{r}$	$\mathbf{r} \in \{y, n, +, -, /\}, \mathbf{c} \in \{Y, N\}$
$*$	Y	$\#$	→	$\#$	$+$	$/$	
$*$	N	$\#$	→	$\#$	$-$	$/$	
$\mathbf{r}$	$-$	$\#$	→	$\#$	$-$	$\mathbf{r}$	$\mathbf{r} \in \{y, n, *\}$
$\mathbf{r}$	$\mathbf{c}$	$\#$	→	$\#$	$\mathbf{r}$	$\mathbf{c}$	$\mathbf{r} \in \{+, -\}, \mathbf{c} \in \{+, -\}$
y	$+$	$\#$	→	$\#$	Y	$*$	
n	$+$	$\#$	→	$\#$	N	$*$	
$/$	$+$	$\#$	→	$\#$	$+$	y	
$/$	$-$	$\#$	→	$\#$	$+$	n	
$*$	$+$	$\#$	→	$\#$	$+$	$*$	

Fig. 3. 24 local rules of the reversible PCA P_{24}, represented by the 10 rule schemes.

We now explain how P_{24} can simulate a CTAG by using the previous example C_0 with an initial string NYY. The initial configuration of P_{24} is set as follows (see the first row of Fig. 4). The string NYY is given in the center parts of some three consecutive cells. The right-part states of these three cells are set to $-$. The states of the cells right to the three cells are set to $(\#, -, -), (\#, Y, -), (\#, Y, -), \cdots$. The production rules (Y, NN, YN) is given by a sequence of the right-part states $y, n, *$, and $-$ in a reverse order, where the sequence $-*$ is used as a delimiter indicating the beginning of a rule. Thus, one cycle of the rules (Y, NN, YN) is represented by the sequence $ny-*nn-*y-*$. We should give infinite copies of the sequence $ny-*nn-*y-*$, since these rules are applied cyclically. The center-part states of all these cells are set to $-$.

−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	N−	Y−	Y−	−−	Y−	Y−	Y−	Y−	Y−
−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−/	Y−	Y−	−−	Y−	Y−	Y−	Y−	Y−
−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	Y/	Y−	−−	Y−	Y−	Y−	Y−	Y−
−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	Y−	Y/	−−	Y−	Y−	Y−	Y−	Y−
−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	Yy	Y−	+n	Y−	Y−	Y−	Y−	Y−
−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	+/	Yy	−+	Yn	Y−	Y−	Y−	Y−
−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−+	Y/	−y	Y+	Yn	Y−	Y−	Y−
−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	Y+	+n	Yy	Y+	Yn	Y−	Y−
−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	Yn	++	Yn	Yy	Y+	Yn	Y−
−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	Yn	N∗	Y+	Yn	Yy	Y+	Yn
−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	+/	Nn	+/	Y+	Yn	Yy	Y+
−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−+	N/	N∗	Y/	Y+	Yn	Yy
−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	N+	N/	+/	Y/	Y+	Yn
−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	Ny	N+	+y	Y/	Y/	Y+
−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	Nn	Ny	++	Yy	Y/	Y/
−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−/	Nn	Y∗	Y+	Yy	Y/
−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	N/	Yn	+/	Y+	Yy
−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	N−	Y/	N∗	Y/	Y+
−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	Ny	Y−	N/	+/	Y/
−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	−/	Yy	N−	+y	Y/
−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	−−	Y/	Ny	−+	Yy
−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	−n	Y−	N/	−y	Y+
−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	−n	Yn	N−	+n	Yy
−∗	−n	−y	−−	−∗	−n	−n	−−	−∗	−y	−−	−∗	−n	−y	−−	−∗	Yn	Nn	−+	Yn

Fig. 4. Simulating a CTAG C_0 with an initial string NYY (in Example 1) by the reversible PCA P_{24} (the state # is indicated by a blank).

We can see the right-part states $y, n, *$, and $-$ act as right-moving signals until they reach the first symbol of a rewritten string.

A rewriting process of C_0 is simulated as below. If the signal $*$ meets the state Y (or N, respectively), which is the head symbol of a rewritten string, then the signal changes Y (N) to the state $+$ $(-)$, and the signal itself becomes / that is sent rightward as a used (garbage) signal. At the next time step, the center-part state $+$ $(-)$ meets the signal $-$, and the former becomes a right-moving signal $+$ $(-)$, and the latter (i.e., $-$) is fixed as a center-part state at this position. The right-moving signal $+$ $(-)$ travels through the rewritten string consisting of Y's and N's, and when it meets the center-part state $+$ or $-$, then it is fixed as a new center-part state, which indicates the last head symbol is Y (N). Note that the old center-part state $+$ $(-)$ is sent rightward as a used signal.

Signals y and n go rightward through the rewritten string consisting Y's and N's until it meets $+$ or $-$. If y (n, respectively) meets $+$, then the signal becomes Y (N) and is fixed at this position (since the last head symbol is Y), and $+$ is shifted to the right by one cell. If y or n meets $-$, then the signal simply continues to travel rightward without being fixed (since the last head symbol is N). Note that all the used and unused informations are sent rightward as garbage signals, because they cannot be deleted by the constraint of reversibility. By the above method, the rewriting process is correctly simulated. Fig. 4 shows how a computation of C_0 is performed in P_{24}. It is easily seen that any CTAG can be simulated in this way.

4 Concluding remarks

In this paper, we proposed a 24-state computation-universal 1-D RCA P_{24} that simulates an arbitrary CTAG on infinite configurations. The basic idea to simulate CTAGs is similar to those in [11, 12]. But here, some states of P_{24} (especially, the center states $Y, +$, and $-$) are designed to play multiple roles of the previous models. By this, the number of states was reduced.

P_{24} was constructed by using the framework of PCAs. An advantage of using a PCA, besides the fact it facilitates to design RCAs, is that the total number of local rules is exactly the same as the number of states, and hence it is much smaller than that of a conventional CA. In fact, P_{24} has a simple local function given by only 10 rule schemes shown in Fig. 3. On the other hand, since the state set of each cell is the direct product of the sets of the three parts, the number of states of a PCA cannot be very small. We have a conjecture that there is a universal 1-D RCA of less than 10 states. However, to obtain such a small universal 1-D RCA, some new framework for designing RCAs should be given.

References

[1] Banks, E.R., Information processing and transmission in cellular automata, PhD Thesis, MIT (1971).

[2] Berlekamp, E., Conway, J., and Guy, R., *Winning Ways for Your Mathematical Plays*, Vol.2, Academic Press, New York (1982).

[3] Codd, E.F., *Cellular Automata*, Academic Press, New York (1968).

[4] Cook, M., Universality in elementary cellular automata, *Complex Systems*, **15**, 1–40 (2004).

[5] Imai, K., and Morita, K., A computation-universal two-dimensional 8-state triangular reversible cellular automaton, *Theoretical Computer Science*, **231**, 181–191 (2000).

[6] Lindgren, K., and Nordahl, M.G., Universal computation in simple one-dimensional cellular automata, *Complex Systems*, **4**, 299–318(1990).

[7] Margolus, N., Physics-like model of computation, *Physica*, **10D**, 81–95 (1984).

[8] Morita, K., and Harao, M., Computation universality of one-dimensional reversible (injective) cellular automata, *Trans. IEICE Japan*, **E72**, 758–762 (1989).

[9] Morita, K., and Ueno, S., Computation-universal models of two-dimensional 16-state reversible cellular automata, *IEICE Trans. Inf. & Syst.*, **E75-D**, 141–147 (1992).

[10] Morita, K., Tojima, Y., Imai, K., and Ogiro, T., Universal computing in reversible and number-conserving two-dimensional cellular spaces, in *Collision-based Computing* (ed. A. Adamatzky), 161–199, Springer-Verlag (2002).

[11] Morita, K., Simple universal one-dimensional reversible cellular automata, *Journal of Cellular Automata*, **2**, 159–166 (2007).

[12] Morita, K., Reversible computing and cellular automata — a survey, *Theoretical Computer Science*, **395**, 101–131 (2008).

[13] von Neumann, J., *Theory of Self-reproducing Automata* (ed. A.W. Burks), The University of Illinois Press, Urbana (1966).

[14] Ollinger, N., and Richard, G., A particular universal cellular automaton, oai:hal.archives-ouvertes.fr:hal-00095821_v2 (2006).

[15] Toffoli, T., Computation and construction universality of reversible cellular automata, *J. Comput. Syst. Sci.*, **15**, 213–231 (1977).

Construction of reversible cellular automata by amalgamations and permutations of states

Juan Carlos Seck Tuoh Mora[1], Manuel González Hernández[1], Harold V. McIntosh[2], and Sergio V. Chapa Vergara[3]

[1] Centro de Investigación Avanzada en Ingeniería Industrial
Universidad Autónoma del Estado de Hidalgo
Carr. Pachuca Tulancingo Km 4.5
Pachuca 42184 Hidalgo, México,
jseck@uaeh.edu.mx,
[2] Centro de Cálculo, Benemérita Universidad Autónoma de Puebla
Apartado Postal 1200, Puebla, México
WWW home page: http://delta.cs.cinvestav.mx/ ~mcintosh/
[3] Departamento de Computación, CINVESTAV-IPN
AV. IPN 2508, México DF 04200, México

Abstract. This paper explains the properties of amalgamations and permutations of states in the matrix representation of reversible one-dimensional cellular automata where both evolution rules have neighborhood size 2 and a Welch index equal to 1. These properties are later used for constructing reversible automata and defining a compact nomenclature to identify them. Some examples are provided.

1 Introduction

Cellular automata are discrete dynamical systems characterized by simple local interactions among their components which are able to produce complex global behaviors, reason why they have been used to simulate natural and artificial systems and for implementing new paradigms of unconventional computing [1] [2]. A classical study in dynamical systems is to understand reversible properties; in cellular automata the initial works in this sense are developed by Moore and Myhill studying Garden-of-Eden configurations [3] [4].

The first computational search for non-trivial reversible automata is developed by Amoroso and Patt [5]; meanwhile relevant theoretical results are obtained by Hedlund [6] and Nasu [7] using topological, combinatorial and graphical results for characterizing the local properties of reversible one-dimensional cellular automata.

These ones have been widely used by others for implementing algorithms to detect and classify reversible one-dimensional cellular automata [8] [9] [10]; in particular Boykett [11] gives an exhaustive enumeration up to 10 states. On the other hand, there is a series of novel results about the characterization of reversible one-dimensional cellular automata by symbolic dynamics tools, which currently have been applied for defining a maximum bound for the inverse minimum mapping in these systems [12] [13] [14] [15] [16].

All the previous manuscripts are the motivation to present a procedure for constructing reversible one-dimensional cellular automata; up to know most of the algorithms calculate candidate evolution rules for achieving reversibility and then they apply an extra procedure for reviewing whether the rules are reversible or not. The original part in this paper is to take the theoretical results in the reversible one-dimensional case with a Welch index equal to 1 in order to provide a general matrix structure for generating only reversible automata without performing an additional revision.

This paper is organized as follows: Sect. 2 exposes the basic concepts of reversible one-dimensional cellular automata with neighborhood size 2 in both evolution rules and a Welch index equal to 1. Section 3 describes the characterization of the matrix representation for reversible automata using amalgamations and permutations of states, Sect. 4 uses this characterization for explaining the general structure of the matrix representing a reversible automaton and with this an explicit construction of such a matrices is given. Section 5 gives examples illustrating the results of the paper and finally concluding remarks are presented.

2 Basic concepts

A cellular automaton $\mathcal{A} = \{s, n, \varphi\}$ is composed by a finite set of states S with $|S| = s$ and a mapping $\varphi : S^n \rightarrow S$ where n is the neighborhood size, φ is the evolution rule and every $w \in S^n$ is a neighborhood in the automaton. The definition of φ can be extended for larger sequences; for $v \in S^m$ with $m \geq n$, $\varphi(v) = w \in S^{m-n+1}$ is the sequence obtained applying φ over all the overlapping neighborhoods in v. With this, for every $w \in S^*$; let $\Lambda(w) = \{v : v \in S^*, \varphi(v) = w\}$ be the set of ancestors of w.

The dynamics of the automaton is discrete in time and space; initially a finite configuration $c^0 : \mathbb{Z}_{m_c} \rightarrow S$ is taken with periodic boundary conditions where c_i^0 is the cell at position $i \; mod \; m_c$ and $\gamma(c_i^0)$ is the state in c_i^0. For a configuration c^t, let $\eta(c_i^t) = \gamma(c_i^t), \gamma(c_{i+1}^t), \ldots, \gamma(c_{i+n-1}^t)$ be the neighborhood specified starting from c_i^t; hence $\varphi(\eta(c_i^t)) = c_i^{t+1}$ which is placed centered below $\eta(c_i^t)$. This process is applied over all the cells in c^t producing a new configuration c^{t+1}, in this way the global dynamics of the automaton is given by the mapping among configurations.

Several papers have shown that every one-dimensional cellular automaton can be simulated by another with neighborhood size 2 grouping states and using a larger set of states [12] [13] [17]. For every sequence $v \in S^{2n-2}$ we have that $\varphi(v) = w \in S^{n-1}$; let T be a new set of states such that $|T| = s^{n-1}$, so a bijection $\beta : T \rightarrow S^{n-1}$ can be defined and with this a new evolution rule $\phi = \beta \circ \varphi \circ \beta^{-1} : T^2 \rightarrow T$ is formed holding the same dynamics that φ and with neighborhood size 2.

A one-dimensional cellular automaton is reversible iff the mapping among configurations is an automorphism; in this case the evolution rule has an inverse φ^{-1} (commonly with different neighborhood size that φ) inducing the invertible dynamics in the automaton. It is clear in the reversible case that the simulation previously described can be applied for both evolution rules taking the largest neighborhood size; so every reversible automaton can be simulated by another where both evolution rules have neighborhood size 2. Saying this, in the rest of this paper only reversible automata with these features are treated.

The evolution rule in a reversible automaton can be described by a matrix M_φ where row and column indices are the states in S and the entry $M_\varphi(a,b) = \varphi(ab)$. This matrix is just representing the well-known de Bruijn diagram associated with the evolution rule [18]. Before presenting the most relevant properties of these systems, useful notation is introduced. For every $w \in S^*$, let:

$$\begin{aligned} \Omega_l(w) &= \{a : a \in S, \text{ there exists } v \in S^* \text{ such that } av \in \Lambda(w)\} \\ \Omega_r(w) &= \{a : a \in S, \text{ there exists } v \in S^* \text{ such that } va \in \Lambda(w)\} \end{aligned} \tag{1}$$

Thus $\Omega_l(w)$ and $\Omega_r(w)$ represent respectively the leftmost and rightmost states in the ancestors of w.

Hedlund [6] and Nasu [7] provide a complete local characterization of reversible one-dimensional cellular automata; relevant results from these works are:

Property 2.1. For every $w \in S^*$, $|\Lambda(w)| = s$.

Property 2.2. For every $w \in S^*$, $\Omega_l(w) = l$ and $\Omega_r(w) = r$ holding that $lr = s$.

Property 2.3. For every pair $w, v \in S^*$, $|\Omega_r(w) \cap \Omega_l(v)| = 1$.

Values l and r are known as Welch indexes, and a reversible automaton is described as $\mathcal{A} = \{s, n, \varphi, l, r\}$.

3 Permutations, definiteness and amalgamations

As it was mentioned before, this work treats only with reversible automata of kind $\mathcal{A} = \{s, 2, \varphi, s, 1\}$ since they have nice properties about the permutation and amalgamation of states which facilitates the analysis; thus the results presented in this manuscript are analogous for the case where $l = 1$ and $r = s$. For a cellular automaton with Welch indices $l = s$ and $r = 1$; in order to hold with Properties 2.1 - 2.3, additional properties are also fulfilled:

Property 3.1. Every state in S appears in one and only one diagonal element in M_φ.

Property 3.2. Every row in M_φ is a permutation of S.

Property 3.3. Every state in S appears in all the rows of M_φ.

Since M_φ is a matrix representation of a de Bruijn diagram; the results established in [7] about the definiteness of such diagrams can be applied into M_φ. For this, let M_φ^2 be the symbolic square of the matrix M_φ, that is; if $M_\varphi(a,b) = s_1$ and $M_\varphi(b,c) = s_2$ then $s_1 s_2 \in M_\varphi^2(a,c)$. For a reversible automaton with neighborhood size 2 in both evolution rules, the following property is given:

Property 3.4. Every $w \in S^2$ appears in the s rows of a single column in M_φ^2.

Property 3.4 says that, the s paths representing each sequence of two states finish into the same node in the de Bruijn diagram. Reversible automaton can be seen as mappings among sequence of symbols which iterations conserve the information of the system, in this way the action of these mappings must be able to produce every possible sequence of states, defining only one ancestor sequence with the same size.

This is related to full shifts in symbolic dynamics, where these systems are able to produce in a unique way any word given by a finite alphabet of symbols. Classically a full shift of s symbols is graphically represented by a single node with s loops, each labeled by a corresponding symbol of the alphabet. A well-known result in symbolic dynamics is that given a finite alphabet, every other labeled digraph able to yield any possible word is conjugated with the full shift; that is, there is a transformation connecting both graphs on either sense [19] [20] [12] [21]. This transformation is performed by splittings and amalgamations, so they can be applied to convert the de Bruijn diagram (and its matrix representation M_φ) into the full shift in the case of a reversible automaton. For this reason, amalgamations are defined over M_φ as follows:

Definition 3.1 (Amalgamation over M_φ). *Let $i_1, \ldots, i_m$ be m identical rows in M_φ. Hence the rows and columns with these indices are replaced by a unique row and column respectively with index $I = \{i_1, \ldots, i_m\}$ conforming a new matrix $M_{\varphi,1}$. For each entry $M_\varphi(i,j)$ the corresponding entry in $M_{\varphi,1}$ is specified as follows:*

- *If $i \notin I$ and $j \notin I$ then $M_{\varphi,1}(i,j) = M_\varphi(i,j)$.*
- *If $i \notin I$ and $j \in I$ then $M_\varphi(i,j) \in M_{\varphi,1}(i,I)$.*
- *If $i \in I$ and $j \notin I$ then $M_\varphi(i,j) \in M_{\varphi,1}(I,j)$.*
- *If $i \in I$ and $j \in I$ then $M_\varphi(i,j) \in M_{\varphi,1}(I,I)$.*

In this way, two rows are amalgamated if the corresponding nodes in the de Bruijn diagram reach the same nodes with the same labeled outgoing edges. Thus, amalgamations are directly associated with definiteness; if M_φ^2 fulfills that all the repetitions of every sequence of states appear in a single column, hence all the corresponding paths of length two in the de Bruijn diagram terminate at the same node. The previous observation implies the following property:

Property 3.5. $M_{\varphi,2}$ is a matrix with a single row and a single column whose unique entry contains all the states in S.

In fact Property 3.5 claims that a reversible automaton with neighborhood size 2 in both evolution rules must carry out that its matrix M_{φ} can be transformed into the full shift after two amalgamations. Given a reversible automaton, the states of the entries in M_{φ} can be permuted and the resulting automaton is still reversible; in fact these permutations correspond to permuting labels in the directed edges of the de Bruijn diagram which is an isomorphism. In this sense, applying a permutation of rows and columns over M_{φ} produces another reversible automaton; this is because this action just implies the permutation of labels in the nodes of the de Bruijn diagram which is also an isomorphism. By properties 3.4 and 3.5 the matrix M_{φ} has $\alpha \geq 2$ equal rows; with this a final property is established:

Property 3.6. In a reversible automaton $\mathcal{A} = \{s, 2, \varphi, s, 1\}$, the entries, rows and columns of M_{φ} can be permuted in order to achieve that:

- $M_{\varphi}(a, a) = a$ for every $a \in S$.
- The final α rows of M_{φ} are equal.

Thus every reversible automaton can be transformed into another one holding with Property 3.6, therefore we only need to construct this kind of reversible automaton to obtain the others. The previous properties are used in the next section for constructing explicitly reversible automata $\mathcal{A} = \{s, 2, \varphi, s, 1\}$.

4 Construction of reversible automata

Property 3.6 gives a lot of structure to the matrix M_{φ} which can be used to allow a explicit construction of reversible cellular automata; first of all there are α identical final rows in M_{φ}, hence the matrix can be divided in four submatrices as follows:

$$M_{\varphi} = \begin{array}{c|c|c} & 1 \;\ldots\; s-\alpha & s-\alpha+1 \;\ldots\; s \\ \hline \begin{matrix}1\\ \vdots\\ s-\alpha\end{matrix} & R_1 & R_2 \\ \hline \begin{matrix}s-\alpha+1\\ \vdots\\ s\end{matrix} & R_3 & R_4 \end{array} \qquad (2)$$

The last α rows (defined by submatrices R_3 and R_4) are identical so by Property 3.6 the submatrix R_4 has the following specification:

$$R_4 = \begin{array}{c|ccc} & s-\alpha+1 & \ldots & s \\ \hline s-\alpha+1 & s-\alpha+1 & \ldots & s \\ \vdots & \vdots & \ddots & \vdots \\ s & s-\alpha+1 & \ldots & s \end{array} \qquad (3)$$

By the same property, rows in submatrix R_3 must be identical, therefore they are defined taking the same permutation $\mu : \{1, \ldots, s-\alpha\} \rightarrow \{1, \ldots, s-\alpha\}$. In this way the submatrix R_3 is established as follows:

$$R_3 = \begin{array}{r} \\ s-\alpha+1 \\ \vdots \\ s \end{array} \begin{array}{c} \begin{array}{ccc} 1 & \ldots & s-\alpha \end{array} \\ \begin{bmatrix} \mu(1) & \ldots & \mu(s-\alpha) \\ \vdots & \ddots & \vdots \\ \mu(1) & \ldots & \mu(s-\alpha) \end{bmatrix} \end{array} \tag{4}$$

In the same sense, every row i in R_1 is defined by a permutation $\pi_i : \{1, \ldots, s-\alpha\} \rightarrow \{1, \ldots, s-\alpha\}$; that is, we may have different permutations in the rows of the submatrix, thus R_1 has a less restrictive structure than the previous ones. However this structure depends on the following characteristics by Properties 3.5 and 3.6:

- $R_1(i,i) = i$ for $1 \leq i \leq s-\alpha$.
- The rows in R_1 must be identical to the rows in R_4 in M_φ or in $M_{\varphi,1}$.

The last restriction assures that M_φ can be amalgamated in at most two steps. Thus the permutations in R_1 depends on the features of the single permutation defining the rows in R_4; since the diagonal elements of R_1 are fixed and ordered, the cyclic representation of μ can be used to specify the rows in R_1. For $i \leq s$, let $\mu_c = c_1, \ldots, c_i$ be the decomposition in cycles of μ, where for $1 \leq j \neq k \leq i$ it is fulfilled that:

- $c_j \subseteq S$
- $\cup_{j=1}^{i} c_j = S$
- $c_j \cap c_k = \emptyset$

Thus every cycle c_j indicates that the corresponding rows in R_1 must be identical; that is, they are specified by the same permutation. In this way for each c_j in μ_c there is a permutation $\pi_j : \{1, \ldots, s-\alpha\} \rightarrow \{1, \ldots, s-\alpha\}$ establishing identical rows in R_1 whose indices are listed by the elements of c_j.

By Property 3.6, for every $i \in c_j$ it is hold that $R_1(i,i) = i$; since the rows enumerated by c_j must be identical, for every $i_1, i_2 \in c_j$ it is fulfilled that $R_1(i_1,i_2) = R_1(i_2,i_2)$. This feature fixes some of the elements in the rows indexed by c_j and the other elements described by π_j can be freely assigned with the restriction that the resultant cycles between π_j and μ must be contained in the cycles of μ_c. More alternatives for R_1 can be achieved joining cycles in μ_c; let μ_g be the cycles produced by grouping elements from μ_c such that at least one cycle in μ_c remains without change and, the cycles between π_j and μ are contained in the ones of μ_g.

Again, every cycle in μ_g indicates that the corresponding rows in R_1 must be specified by the same permutation. Joining cycles in this way preserves the amalgamation in two steps of M_φ because the original cycles in μ_c are contained in the ones from μ_g, therefore the latter specifies a larger number of identical rows in R_1. From here, for μ it is said that μ_c is its minimum cyclic representation and μ_g is equal or contains the cycles in μ_c, so μ_g shall be used in order to define the permutations conforming the rows in R_1.

For $i \in \{1, \dots, s-\alpha\}$, $j \leq s$ such that $\mu_g = c_1, \dots, c_j$ and $1 \leq k \leq j$, let us define $\rho(i) = k$ such that $i \in c_k \in \mu_g$, with this, the permutations conforming the submatrix R_1 can be detailed in the following way:

$$R_1 = \begin{array}{c c} & \begin{array}{ccc} 1 & \dots & s-\alpha \end{array} \\ \begin{array}{c} 1 \\ \vdots \\ s-\alpha \end{array} & \left[\begin{array}{ccc} \pi_{\rho(1)}(1) & \dots & \pi_{\rho(1)}(s-\alpha) \\ \vdots & \ddots & \vdots \\ \pi_{\rho(s-\alpha)}(1) & \dots & \pi_{\rho(s-\alpha)}(s-\alpha) \end{array}\right] \end{array} \tag{5}$$

Finally, each permutation π_j has associated a permutation $\sigma_j : \{s-\alpha+1, \dots, s\} \rightarrow \{s-\alpha+1, \dots, s\}$ to complete the submatrix R_2; so each π_j may have different permutations σ_j assigned in R_2. With this, the structure of this submatrix has the following description:

$$R_2 = \begin{array}{c c} & \begin{array}{ccc} s-\alpha+1 & \dots & s \end{array} \\ \begin{array}{c} 1 \\ \vdots \\ s-\alpha \end{array} & \left[\begin{array}{ccc} \sigma_{\rho(1)}(s-\alpha+1) & \dots & \sigma_{\rho(1)}(s) \\ \vdots & \ddots & \vdots \\ \sigma_{\rho(s-\alpha)}(s-\alpha+1) & \dots & \sigma_{\rho(s-\alpha)}(s) \end{array}\right] \end{array} \tag{6}$$

With the preceding structures delineating the submatrices composing M_φ, we can propose a way for numbering a reversible automaton $\mathcal{A} = \{s, 2, \varphi, s, 1\}$ as $(\alpha, \mu, [\mu_g], [(\pi_1, \sigma_1), \dots, (\pi_j, \sigma_j)])$; where:

- α is the number of identical rows in M_φ.
- μ is the factoradic number for the permutation defining R_3.
- $\mu_g = c_1, \dots, c_j$ is the set of grouped cycles from μ_c.
- π_k is the factoradic number for the permutation in R_1 associated with c_k for $1 \leq k \leq j$.
- σ_k is the factoradic number for the permutation in R_2 associated with π_k.

5 Examples

This section applies the previous results in a reversible automaton $\mathcal{A} = \{6, 2, \varphi, 6, 1\}$; taking $\alpha = 2$ the matrix M_φ has initially the following form.

$$M_\varphi = \begin{array}{c c} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \left[\begin{array}{cccc|cc} 1 & & & & & \\ & 2 & & & & \\ & & 3 & & & \\ & & & 4 & & \\ \hline & & & & 5 & 6 \\ & & & & 5 & 6 \end{array}\right] \end{array}$$

For μ there are 4! possibilities, taking $\mu = 14 = (3214)$ the matrix M_φ changes as follows:

$$M_\varphi = \begin{array}{c|cccc|cc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & & & & & \\
2 & & 2 & & & & \\
3 & & & 3 & & & \\
4 & & & & 4 & & \\ \cline{2-7}
5 & 3 & 2 & 1 & 4 & 5 & 6 \\
6 & 3 & 2 & 1 & 4 & 5 & 6
\end{array}$$

In this way $\mu_c = (13)(2)(4)$, therefore at least rows $1, 3$ must be equal in M_φ to conserve the amalgamation in two steps.

$$M_\varphi = \begin{array}{c|cccc|cc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & & 3 & & & \\
2 & & 2 & & & & \\
3 & 1 & & 3 & & & \\
4 & & & & 4 & & \\ \cline{2-7}
5 & 3 & 2 & 1 & 4 & 5 & 6 \\
6 & 3 & 2 & 1 & 4 & 5 & 6
\end{array}$$

Taking $\mu_g = \mu_c$ we have to specify three permutations π for every cycle in μ_g such that the cycle between π and μ is contained in μ_g. Particularly, for the cycle (13) the unique possibility is $\pi_1 = 0 = (24)$, for the cycle (2) the possibilities for π_2 are $0 = (134)$ and $2 = (314)$ and for the cycle (4) the possibilities for π_3 are $0 = (123)$ and $5 = (321)$. Let us assign $\pi_1 = 0$, $\pi_2 = 2$ and $\pi_3 = 0$, with this M_φ acquires the next form:

$$M_\varphi = \begin{array}{c|cccc|cc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & & \\
2 & 3 & 2 & 1 & 4 & & \\
3 & 1 & 2 & 3 & 4 & & \\
4 & 1 & 2 & 3 & 4 & & \\ \cline{2-7}
5 & 3 & 2 & 1 & 4 & 5 & 6 \\
6 & 3 & 2 & 1 & 4 & 5 & 6
\end{array}$$

Finally for every permutation π the associated permutation σ may be $0 = (56)$ and $1 = (65)$; thus a possible specification is $\sigma_1 = 1$, $\sigma_2 = 0$, $\sigma_3 = 0$ and, M_φ is completed.

$$M_\varphi = \begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & 6 & 5 \\
2 & 3 & 2 & 1 & 4 & 5 & 6 \\
3 & 1 & 2 & 3 & 4 & 6 & 5 \\
4 & 1 & 2 & 3 & 4 & 5 & 6 \\
5 & 3 & 2 & 1 & 4 & 5 & 6 \\
6 & 3 & 2 & 1 & 4 & 5 & 6
\end{array}$$

With this, the automaton is labeled as $\{2, 14, [(13)(2)(4)], [(0, 1), (2, 0),(0, 0)]\}$. In order to prove that this automaton is reversible, $M_{\varphi,2}$ must be identical with the full shift; this is fulfilled making the amalgamations over M_φ:

$$M_{\varphi,1} = \begin{array}{r} \\ (1,3) \\ (2) \\ (4) \\ (5,6) \end{array} \!\!\begin{array}{c} \begin{array}{cccc} (1,3) & (2) & (4) & (5,6) \end{array} \\ \left[\begin{array}{cccc} 1,3 & 2 & 4 & 6,5 \\ 3,1 & 2 & 4 & 5,6 \\ 1,3 & 2 & 4 & 5,6 \\ 3,1 & 2 & 4 & 5,6 \end{array}\right] \end{array} \qquad M_{\varphi,2} = \begin{array}{cc} & (1,3,2,4,5,6) \\ (1,3,2,4,5,6) & \left[\,1,3,2,4,6,5\,\right] \end{array}$$

Let us take now $\mu_g = (13)(24)$, here the cycle (13) from μ_c is conserved according to the specification of R_1 explained in the previous section; hence the matrix M_φ has initially the following arrangement:

$$M_\varphi = \begin{array}{c|cccc|cc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 1 & & 3 & & & \\ 2 & & 2 & & 4 & & \\ 3 & 1 & & 3 & & & \\ 4 & & 2 & & 4 & & \\ \cline{2-7} 5 & 3 & 2 & 1 & 4 & 5 & 6 \\ 6 & 3 & 2 & 1 & 4 & 5 & 6 \end{array}$$

In this way we must have two permutations π for the cycles in μ_g such that every cycle between π and μ is contained in μ_g. For the cycle (13) the possible permutations for π_1 are $0 = (24)$ and $1 = (42)$ and for the cycle (24) the possibilities for π_2 are $0 = (13)$ and $1 = (31)$. Let us assign $\pi_1 = 1$, $\pi_2 = 1$; at last, for every π the permutation σ may be $0 = (56)$ and $1 = (65)$. Let us take $\sigma_1 = 0$ and $\sigma_2 = 1$, with this the automaton is labeled as $\{2, 14, [(13)(24)], [(1, 0), (1, 1)]\}$ and M_φ acquires the next form:

$$M_\varphi = \begin{array}{c|cccccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 1 & 4 & 3 & 2 & 5 & 6 \\ 2 & 3 & 2 & 1 & 4 & 6 & 5 \\ 3 & 1 & 4 & 3 & 2 & 5 & 6 \\ 4 & 3 & 2 & 1 & 4 & 6 & 5 \\ 5 & 3 & 2 & 1 & 4 & 5 & 6 \\ 6 & 3 & 2 & 1 & 4 & 5 & 6 \end{array}$$

Again, to prove that this automaton is reversible, $M_{\varphi,2}$ must be identical with the full shift; this is obtained in the next amalgamations:

$$M_{\varphi,1} = \begin{array}{r} \\ (1,3) \\ (2,4) \\ (5,6) \end{array} \!\!\begin{array}{c} \begin{array}{ccc} (1,3) & (2,4) & (5,6) \end{array} \\ \left[\begin{array}{ccc} 1,3 & 4,2 & 5,6 \\ 3,1 & 2,4 & 6,5 \\ 3,1 & 2,4 & 5,6 \end{array}\right] \end{array} \qquad M_{\varphi,2} = \begin{array}{cc} & (1,3,2,4,5,6) \\ (1,3,2,4,5,6) & \left[\,1,3,4,2,5,6\,\right] \end{array}$$

6 Conclusions

This paper has shown that reversible automata with a Welch index 1 can be completely generated by constructing the corresponding matrices M_φ by means of permutations of states, carrying out that an amalgamation in two steps transforms the matrices into the full shift. Besides the factoradic representation of permutations allows to have a compact nomenclature for identifying every reversible automaton.

These results can be used for implementing algorithms for computing reversible automata or for only enumerating them without an explicit construction; in the latter case the procedure just must count all the valid grouping of cycles for μ_g and in each one of them, enumerate as well all the possible right selections for permutations π and σ. This task may be resolved using simplifications and adaptations of Stirling numbers.

Of course, the most interesting project is to extend these results to all kind of reversible one-dimensional cellular automata; that is, taking into account as well the cases with both Welch indices different from one. In these ones there must be considered amalgamations of rows and columns in the matrices M_φ, where these elements have their own properties but are not anymore permutations from the set of states in the reversible automaton.

References

[1] Chopard, B., Droz, M., eds.: Cellular Automata Modeling of Physical Systems. Cambridge University Press (1998)

[2] Adamatzky, A., ed.: Collision-Based Computing. Springer (2002)

[3] Myhill, J.: The converse of Moore's Garden-of-Eden theorem. Proceedings of the American Mathematical Society **14** (1963) 685–686

[4] Moore, E.F.: Machine models of self-reproduction. In: Essays on Cellular Automata. University of Illinois Press (1970)

[5] Amoroso, S., Patt, Y.: Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. Journal of Computer and System Sciences **6** (1972) 448–464

[6] Hedlund, G.A.: Endomorphisms and automorphisms of the shift dynamical system. Mathematical Systems Theory **3** (1969) 320–375

[7] Nasu, M.: Local maps inducing surjective global maps of one dimensional tesellation automata. Mathematical Systems Theory **11** (1978) 327–351

[8] Hillman, D.: The structure of reversible one-dimensional cellular automata. Physica D **52** (1991) 277–292

[9] Moraal, H.: Graph-theoretical characterization of invertible cellular automata. Physica D **141** (2000) 1–18

[10] Seck-Tuoh, J.C., Juárez, G., Chapa, S.V., McIntosh, H.V.: Procedures for calculating reversible one-dimensional cellular automata. Physica D **202** (2005) 134–141

[11] Boykett, T.: Efficient exhasutive enumeration of reversible one dimensional cellular automata. http://verdi.algebra.uni-linz.ac.at/˜tim (1997)

[12] Seck-Tuoh, J.C., Chapa, S.V., González, M.: Extensions in reversible one-dimensional cellular automata are equivalent with the full shift. International Journal of Modern Physics C **14** (2003) 1143–1160

[13] Czeizler, E., Kari, J.: A tight linear bound on the neighborhood of inverse cellular automata. Proceedings of ICALP, Lecture Notes in Computer Science 3580 (2005) 410–420

[14] Czeizler, E., Kari, J.: On testing the equality of sofic systems. Internal Proceedings of XIth Mons Days Of Theoretical Computer Science (2006)

[15] Czeizler, E., Kari, J.: A tight linear bound on the synchronization delay of bijective automata. To appear in Theoretical Computer Science (2007)

[16] Seck-Tuoh, J.C., Pérez-Lechuga, G., McIntosh, H.V., Juŕez, G., Chapa-Vergara, S.V.: Welch diagrams and out-amalgamations in reversible one-dimensional cellular automata. Submitted to Theoretical Computer Science (2008)

[17] Boykett, T., Kari, J., Taati, S.: Conservation laws in rectangular ca. To appear in Journal of Cellular Automata (2006)

[18] McIntosh, H.V.: Linear cellular automata via de bruijn diagrams. http:// delta. cs. cinvestav. mx/ ˜mcintosh (1991)

[19] Lind, D., Marcus, B.: An Introduction to Symbolic Dynamics and Coding. Cambridge University Press, Cambridge (1995)

[20] Kitchens, B.P.: Symbolic Dynamics One-sided Two-sided and Countable Markov Shifts. Springer-Verlag (1998)

[21] Seck-Tuoh, J.C., Juárez, G., McIntosh, H.V.: The inverse behavior of a reversible one-dimensional cellular automaton obtained by a single welch diagram. Journal of Cellular Automata **1** (2006) 25–39

Encryption using cellular automata chain-rules

Andrew Wuensche[1,2]

[1] Discrete Dynamics Lab. (www.ddlab.org)
[2] University of Sussex, United Kingdom

Abstract. Chain rules are maximally chaotic CA rules that can be constructed at random to provide a huge number of encryption keys — where the the CA is run backwards to encrypt, forwards to decrypt. The methods are based on the reverse algorithm and the Z-parameter [5].

1 The CA reverse algorithm and basins of attraction

In the simplest cellular automata [4], each cell in a ring of cells updates its value (0,1) as a function of the values of its k neighbours. All cells update synchronously — in parallel, in discrete time-steps, moving through a deterministic forward trajectory. Each "state" of the ring, a bit string, has one successor, but may have multiple or zero predecessors.

A book, "The Global Dynamics of Cellular Automata" [5] published in 1992 introduced a reverse algorithm for finding the pre-images (predecessors) of states for any finite 1d binary CA with periodic boundary conditions, which made it possible to reveal the precise topology of "basins of attraction" for the first time — represented by state transition graphs — states linked into trees rooted on attractor cycles, which could be drawn automatically, as in Fig. 1. The software was attached to the book on a floppy disk — the origin of what later became DDLab [12].

As state-space necessarily includes every possible piece of information encoded within the size of its string, including excerpts from Shakespeare, copies of the Mona Lisa, and one's own thumb print, and given that each unique string is linked somewhere within the graph according to a dynamical rule, this immediately suggested that a string with some relevant information could be recovered from another string linked to it in some remote location in the graph, for example by running backwards from string A (the information) to arrive after a number of time steps at string B (the encryption), then running forwards from B back to A to decrypt (or the method could be reversed) — so here was new approach to encryption where the rule is the encryption key.

Gutowitz patented analogous methods using dynamical systems, CA in particular [2], but these are different from the methods I will describe, where its crucial to distinguish a type of CA rule were the graph linking state-space has the appropriate topology to allows efficient encryption/decryption.

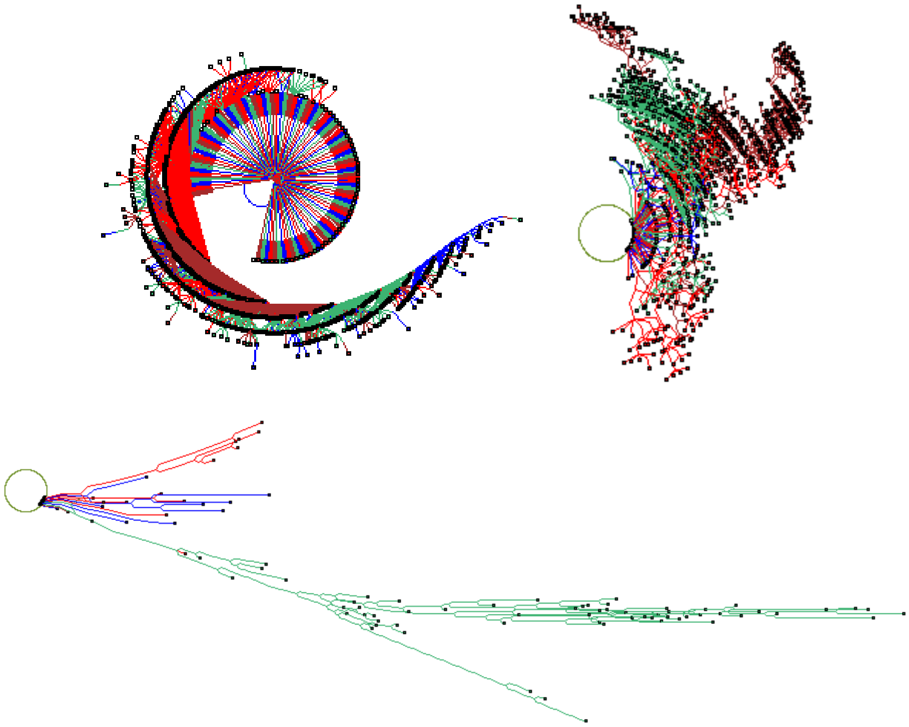


Fig. 1. Three basins of attraction with contrasting topology, n=15, k=3. The direction of time flows inward towards the attractor, then clockwise. One complete set of equivalent trees is shown in each case, and just the Garden-of-Eden (leaf) states are shown as nodes. Data for each is provided as follows: attractor period=p, volume=v, leaf density=d, longest transient=t, max in-degree=P_{max}.
$topleft$: rule 250, Z_{left}=0.5, Z_{right}=0.5, too convergent for encryption, p=1, v=32767, d=0.859, t=14, P_{max}=1364,.
topright: rule 110, Z_{left}=0.75, Z_{right}=0.625, too convergent for encryption, p=295, v=10885, d=0.55, t=39, P_{max}=30,.
bottom: rule 30, a chain rule, Z_{left}=0.5, Z_{right}=1, OK for encryption, p=1455, v=30375, d=0.042, t=321, P_{max}=2,

2 The Z-parameter

Many fascinating insights emerged during the genesis of [5], among them a refinement of Langton's λ-parameter [3]. This "Z-parameter" arose directly from the reverse algorithm, which computed the next unknown bit (say, from left to the right, giving Z_{left}) of a partly known pre-image, or conversely from right to left. The Z-parameter, by analysing just the lookup-table (the CA rule) gave the probability that this next bit was uniquely determined, and being a probability the value of Z_{left} ranged between 0 and 1. The converse direction gave Z_{right}, and the greater of the two was deemed to be the final Z-parameter.

Z did a good job of predicting the bushiness or branchiness of subtrees in basins of attraction — their typical in-degree (or degree of pre-imaging), which related to the density of end states without pre-images (Garden-of-Eden states) but lets call them "leaves" for short. A branchier tree (low Z) has more leaves, and shorter branches (transients) because all those pre-images and leaves use up a finite state-space. Conversely, sparsely branching trees (high Z) have fewer leaves, and longer branches.

Figure 1 gives three examples with contrasting topology.

Low Z, a low probability that the next bit was determined, meant the next bit would probably be both 0 and 1, equally valid, so more pre-images and branchiness, or that there was no valid next bit, more leaves. High Z, a high probability that the next bit was determined, meant it would probably be either 0 or 1, not both, so fewer pre-images, less branchiness, but more chance of a valid pre-image, so fewer leaves.

This nicely tied in with the behaviour of CA when run forward. Low Z, high branchiness results in ordered dynamics. High Z, low branchiness, results in disordered dynamics (chaos), behaviour that could be recognised subjectively, but also by various objective measures, in particular "input entropy" [6, 7]. The entropy stabilises for both order (at a low level) and chaos (at a high level); entropy that does not stabilise but exhibits variance over time is a signature of complexity, with intermediate Z.

3 Limited pre-image rules

Rules in general, even with high values of $Z<1$, can generate huge numbers of pre-images from typical states, which would slow down the reverse algorithm. A branchy (convergent) graph topology has many leaves. As strings become larger, the leaf density increases taking up almost all of state-space, as in Fig. 2 (rule 250).

These leaf states cannot be encrypted by running backward. The alternative, running forwards to encrypt, then backwards to decrypt, poses the problem of selecting the correct path out of a multitude of pre-image branches at each backward time-step. Running forward continually loses information on where you came from; CA are dissipative dynamical systems.

But there is an solution! When $Z=1$, however large the size of the lattice, the number of pre-images of any state is strictly limited. These $Z=1$ rules come in

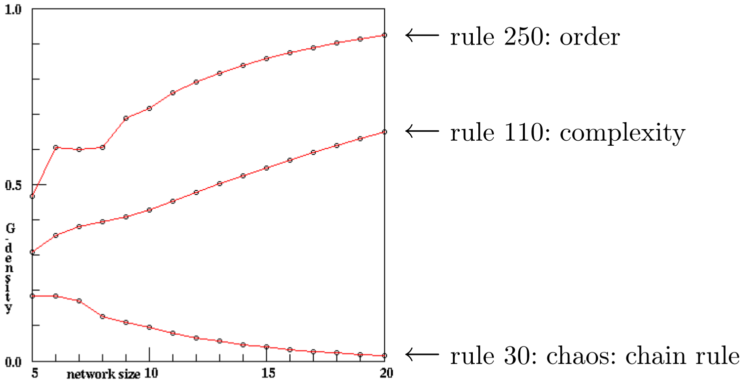


Fig. 2. A plot of leaf (Garden-of-Eden) density with increasing system size, n= 5 to 20, for the three rules in Fig. 1. The measures are for the basin of attraction field, so the entire state-space. For rule-space in general, leaf density increases with greater n, but for chain rules leaf density decreases.

two types [5] as follows, (note that k is the effective-k, some rules have redundant inputs [5]),

- "two-way limited pre-image rules": Z_{left} and Z_{right} both equal one, where the in-degree must be either 2^{k-1} or zero,
- "one-way limited pre-image rules": Z_{left}=1 or Z_{right}=1, but not both, where the in-degree must be less than 2^{k-1}.

Limited pre-imaging (in-degree) appears to produce a promising topology on which to implement encryption, because we would usually need a long string to encode information, but the "two-way" Z=1 rules still suffer from some of the same problems as rules in general, too branchy and a high proportion of leaves.

"One-way" Z=1 rules on the other hand, seem to provide the ideal graph topology. They have an unexpected and not fully understood property: that although the maximum number of pre-images (P_{max}) of a state must be less than 2^{k-1}, experiment shows that the actual number is usually much less, and decreases as the system size increases; consequently the leaf-density also decreases as in Fig. 2 ($rule30$).

For large strings of 1000+, P_{max}=1, except for very rare states where the in-degree=2 in transients, or where transients joint the attractor cycle, which may be extremely long. However, this branching is not a problem when running forward to decrypt, because forward paths converge, and the original pattern will be found. Because the vast majority of state-space occurs in long chains, I renamed the "one-way limited pre-image rules" — "chain-rules".

In Figs. 3 and 4, where k=7, P_{max} must be less than 2^6=64, but the basin of attraction field (for a chain rule constructed at random) has P_{max}=5 and usually less, as shown in the in-degree histogram Figs. 3 (*right*), though there is a small basin where P_{max}=18.

As n increases P_{max} decreases. In Fig. 5 where n=400, P_{max}=2, but 97% of states have an in-degree of one. As n increases further, in-degrees of 2, and leaves, become exceedingly rare, becoming vanishingly small in the limit.

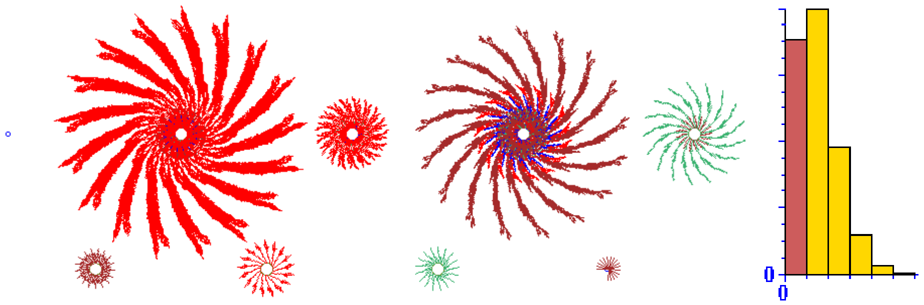


Fig. 3. *left*: The basin of attraction field of a chain rule, showing all 9 basins of attraction (state transition graphs) for the k=7 rule (hex)879ac92e2b44774b786536d1d4bb88b41d. Note there is a tiny attractor (top left) consisting of just one state, all-0s; the last basin (bottom right) has the all-1s point attractor. The chain rule (Z_{left}=0.59, Z_{right}=1) was constructed at random. The string length n=17, state-space=2^{17}=131072, leaf density=0.345, P_m for each basin of the 9 basins is [1,5,5,5,3,4,4,3,18].
right: The in-degree histogram, from 0 to 5, showing in-degree=1 as the most abundant.

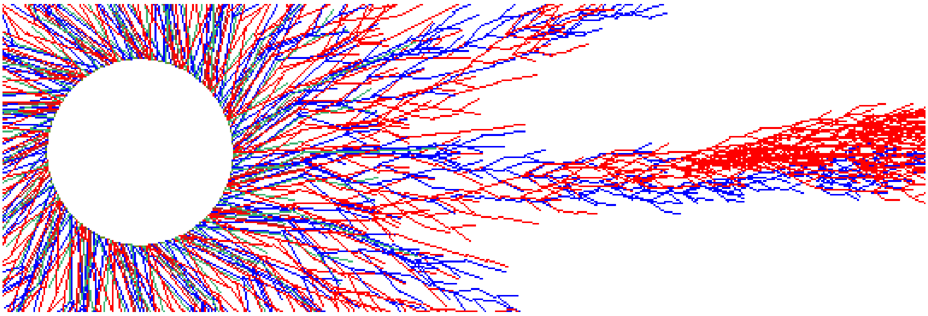


Fig. 4. A detail of the largest basin in Fig. 3, attractor period=357, basin volume=91868 70.1% of state-space, leaf density=0.345, max levels=119, P_m=5.

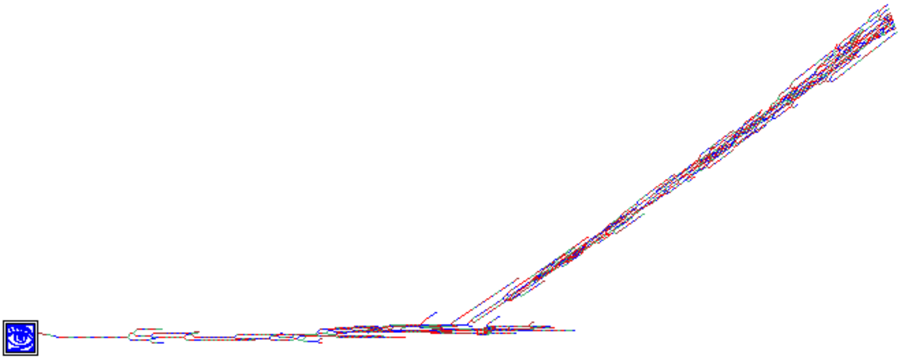


Fig. 5. The subtree of a chain rule, n=400, where the root state is shown in 2d (20×20), with the same chain rule as in Figs. 3 and 4. Backwards iteration was stopped after 400 time-steps. The subtree has 3346 states including the root state. There are 109 leaves (leaf density = 0.0326). P_{max}=2 and the density of these branching states is 0.035.

4 Constructing chain rules at random

The procedure for finding the Z-parameter [5, 6, 7] (its first approximation) from a rule-table (lookup-table) is as follows: Consider pairs of neighbourhoods that differ only by their last right bit, so the k-1 bits on the left are the same. Then look at the outputs of these pairs of neighbourhoods in the rule-table to see if they are the same (00 or 11) or different (01 or 10). Z_{left} is the fraction of *different* pairs in the look-up table. If all the pairs are different then Z_{left}=1. Z_{right} is given by the converse procedure.

There are refinements if effective-k in parts of the rule-table is less than k, but we will not bother with that because the pairs procedure gives the most chaotic dynamics.

To assign a chain rule at random, first pick Z_{left} or Z_{right} at random, then randomly assign different pairs of outputs (01 or 10) to the pairs of neighbourhoods defined above. Check the Z-parameter (with the full algorithm). If Z_{left}=1 or Z_{right}=1 but not both, we have a chain rule.

>From experiment, the lesser value should not be too low, 0.5 or more [9]. This is to avoid a gradual lead-in and lead-out of structure in the space-time pattern when decrypting. Ideally the message, picture, or information, should pop out suddenly from a stream of chaos then rescramble quickly back into chaos, as in Fig. 7. This is accompanied by a lowering blip in the high input-entropy, the duration of the blip needs to be minimised to best “hide” the message.

DDLab [12] can assign a chain at random as above, instantaneously, with a key press, see the DDLab manual [9], section 16.7 and elsewhere.

5 How many chain-rules?

How many chain-rules, C, are there in a rule-space S $= 2^{2^k}$?

The number of ways (say Z_{left}=1) "pairs" can be assigned is $2^{2^{k-1}} = \sqrt[2]{S}$. Adding the same for Z_{left}=1, the number of chain rules C=$2(2^{2^{k-1}})$, but we must subtract the number of rules where both Z_{left}=1 and Z_{right}=1, which is about $2^{2^{k-2}}$, because "pairs" need to be assigned to half of the rule-table, and their compliment to the other half. Subtracting also cases where the lesser value is less than 0.5, a round estimate for the number of acceptable chain rules is $2^{2^{k-1}}$, or the square root of rule-space.

This is sufficiently large to provide an inexhaustible supply of keys, for example for k=5: 2^{16}, k=6: 2^{32}, k=7 :2^{64}, k=8: 2^{128} etc. A chain-rule constructed randomly in DDLab will very probably be a unique key.

6 Encryption/decryption with chain rules

The CA reverse algorithm is especially efficient for chain rules, because the rules-tables are composed purely of "deterministic permutations" — they lack the "ambiguous permutations" that can slow down the reverse algorithm [5].

Many experiments have confirmed that chain rules make basin of attraction topologies that have the necessary properties for encryption. Nearly all states have predecessors and are embedded deeply within long chain-like chaotic transients.

There will still be leaves, and states close to the leaves, patterns that could not be encrypted by that particular chain rule because a backwards trajectory would just stop prematurely. However, for big binary systems, like 1600 as in Figs. 6 and 7, the state-space is so huge, 2^{1600}, that to stumble on an unencryptable state would be very unlikely, but if it were to happen, simply construct a different chain rule.

Encryption/decryption has been available in DDLab since about 1998. To encrypt, select a chain rule (and save it). Select a large enough 1d lattice (which can be shown in 2d). Select the information to be encrypted by loading an ASCII file (for text), or a DDLab image file as the seed, or create a picture with DDLab's drawing function. Select a subtree, and set it to stop after say 20 backwards time-steps. The subtree is unlikely to branch, but if it does, no worries. Save the (encrypted) state reached. Fig. 6 and 8 show examples.

To decrypt, reset DDLab to run forward. Keep the same rule or load it. Load the encrypted state as the initial state. If decrypting a picture, set the presentation for 2d. Run forward, which can be done with successive key-press to see each time-step at leisure. At the 20th time-step, the image will pop out of the chaos. To fully observe the transition, continue stepping forward until the pattern returns to chaos. There will be some degree of ordering/disordering on either side, as in Figs. 7 and 9.

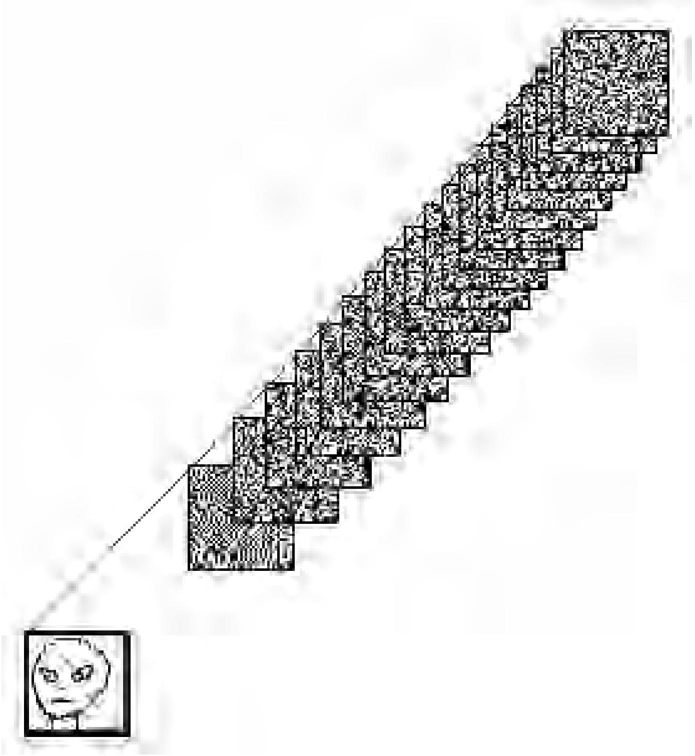


Fig. 6. A 1d pattern is displayed in 2d (n=1600, 40×40); the "alien" seed was drawn with the drawing function in DDLab. The seed could also be an ASCII file, or any other form of information. With a k=7 chain rule constructed at random, and the alien as the root state, a subtree was generated with the CA reverse algorithm; note that the subtree did not branch, and branching is highly unlikely to occur. The subtree was set to stop after 20 backward time-steps. The state reached is the encryption. This figure was taken from [8, 10].

7 Generalising the methods to multi-value

In 2003, all functions and algorithms in DDLab were generalised from binary, v=2 (0,1), to multi-value. The "value-range" v (number of colours) can be set from 2 to 8, i.e. v=3 (0,1,2), v=4 (0,1,2,3), up to v=8 (0,1,2,..7). This included the reverse algorithm, the Z-parameter, chain-rules, and encryption. Details of the new algorithms will be written up at a later date.

Any available value-range can be used for encryption, but for efficiency's sake, not to waste bits, v=2, v=4 or v=8 are preferable.

The examples in Figs. 8 and 9 shows the encryption of a portrait, with v=8, k=4, on a 88×88 lattice (n=7744), but as v=8, the size of the binary string encoding the lattice is 61052. The v=8 chain rule was constructed at random;

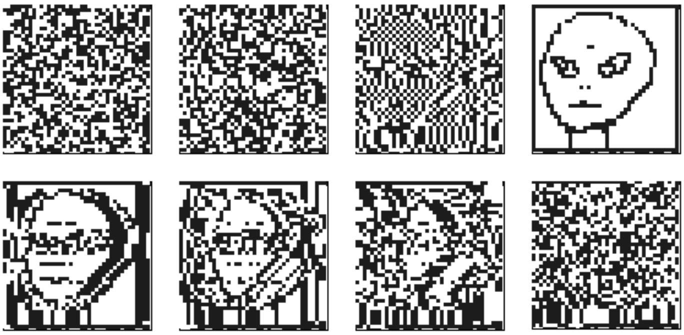


Fig. 7. To decrypt, starting from the encrypted state in Fig. 6 (n=1600, 40×40), the CA with the same rule was run forward by 20 time steps, the same number that was run backward, to recover the original image or bit-string. This figure shows time steps 17 to 25 to illustrate how the "alien" image was scrambled both before and after time step 20. This figure was taken from [8, 10].

Z_{left}=0.4, and Z_{right}=1. When decrypting, the portrait pops out suddenly from chaos, but it takes about 50 time-steps to fully restore chaos. This is because k=4 is a small neighbourhood, and the chaotic pattern moves into areas of order at its "speed of light", set by k.

8 Possible drawbacks

Here are some possible drawbacks of the encryption method.

Chain rules usually result in attractor cycles with very long periods, though this is relative — the fraction of state-space made up of attractor cycles is probably small. If a state to be encrypted happens to be on an attractor cycle, running it backward may arrive at a point where a transient joins the attractor. In this case the backwards trajectory will branch.

Note also that there is an effect called "rotational symmetry" (and also "bilateral symmetry") that is inescapable in classical CA, where states with greater symmetry must be downstream of states with lesser symmetry, or with none [5]. This means that the uniform states, all-0s and all-1s, must be downstream of all other states in the dynamics, and the states like 010101.. downstream of states like 00110011.., which are downstream of the rest, etc. However, these highly ordered states hold little or no information, so are irrelevant for encryption.

A consequent effect is that "rotational symmetry" must stay constant in an attractor cycle, so in binary systems each uniform state must be one of the following: a point attractor; a transient state leading directly to the other's point attractor; part of a 2-state attractor with the other. In multi-value networks things get more complicated.

The key itself (the chain-rule) must be transmitted somehow — and with perfect accuracy. Noise in transmission of the encryption will spread during

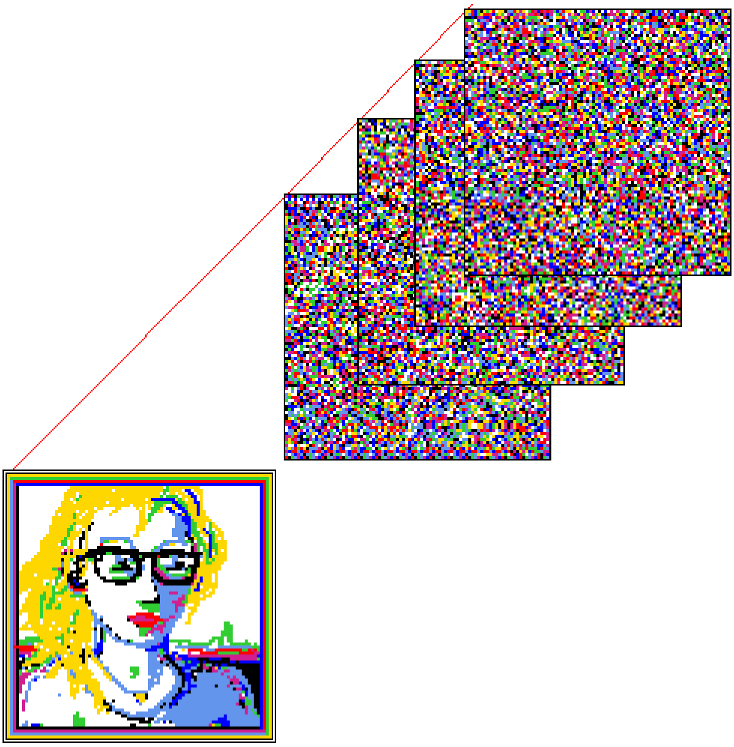


Fig. 8. A 1D pattern is displayed in 22 (n=7744, 88×88). The "portrait" was drawn with the drawing function in DDLab. With a v=8, k=4 chain rule constructed at random, and the portrait as the root state, a subtree was generated with the CA reverse algorithm. The subtree was set to stop after 5 backward time-steps. The state reached is the encryption.

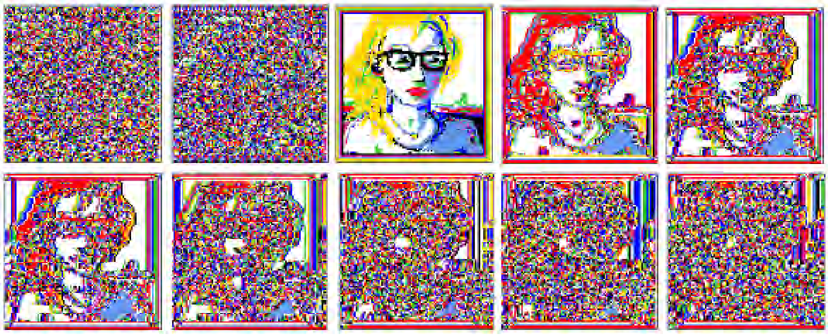


Fig. 9. To decrypt, starting from the encrypted state in Fig. 8 (n=7744, 88×88), the CA with the same rule was run forward to recover the original image. This figure shows time steps -2 to +7.

decryption, but only at the "speed of light" depending on k and the number of forward time-steps, so the process is relatively robust to this extent.

9 A bit of history

The encryption method described in this paper relies on a number of ideas first published in 1992 [5] and developed from work started in about 1988. The reverse algorithm for running 1d CA backwards made it possible to invent the Z-parameter, to reveal the topology of basins of attraction, and begin to study and understand these objects - how they relate to rule-space.

The "limited pre-image rules" in [5] were renamed "chain-rules" for brevity in about 1998, but the principle and possibility of encrypting by running CA backward, decrypting by running forward, using "limited pre-image rules", was well know to the authors of [5] at the time of writing. The method was made to work within DDLab in about 1998.

I've often described this encryption method in talks and lectures, including live demos of encryption with DDLab in many venues in a number of countries. The first time for a big audience was at the SFI summer school in Santa Fe in 1999. Up till now I have written only brief accounts specifically on this encryption method, both published [9, 11] and unpublished [8, 10].

I'm relating this bit of history because I have just now read a new paper by Gina M. B. Oliviera and others [1], sent to me for review, presenting their independent discovery of the same encryption method; except they used a genetic algorithm to evolve the rule-tables (instead of constructing them) to produce the Z-parameter property: Z_{left}=1 or Z_{right}=1 but not both (with some refinments) — the exact definition of chain rules.

Other than spurring me on to write this paper (for which I am most grateful, and also for their citations) I must say that as far as the science goes, I have not been influenced by their paper in any way.

10 Conclusions

I have described a method of encryption where the key is a chain-rule, a special type of maximally chaotic 1d CA rule.

Information is encrypted by using a key to run the CA backward in time. A secret message can be transmitted openly. The receiver has the same key, and uses it to decipher the message by running the CA forward in time by an equal number of steps. Anyone could construct their own unique key instantaneously from a virtually unlimited source — its size is about the square root of rule-space.

What is important to someone trying to crack an intercepted encrypted message, with DDLab available? The key itself is vital; data on the CA, its neighbourhood k, and value-range v, is important — not obvious from the key itself. The number of time-steps are useful, to know when the forward run should stop.

Suppose both the raw message and the encryption where known, could the key be deduced? I do not see how if the two are separated by a number of time-steps, without knowing the intervening steps.

In other security measures, the key itself could be encrypted. A message could be doubly or multiply encrypted with more than one key.

Although these methods are available in DDLab, dedicated software and hardware could be developed for the whole procedure to be fast, hidden and automatic, and also to handle data streams in real time.

References

[Note] For publications by A.Wuensche refer to `www.cogs.susx.ac.uk/users/andywu/publications.html` where most are available for download.

[1] Oliveira, G.M.B, H.B. Macódo, A.B.A. Branquinho, and M.J.L. Lima., A cryptographic model based on the pre-image calculus of cellular automata, to appear in the proceeding Automata-2008.

[2] Gutowitz, H., Method and Apparatus for Encryption, Decryption, and Authentication using Dynamical Systems, U.S. patent application (1992)

[3] Langton, C.G., Computation at the edge of chaos: Phase transitions and emergent computation", Physica D, 42, 12-37, 1990.

[4] Wolfram, S., Statistical mechanics of cellular automata, Reviews of Modern Physics, vol 55, 601–644, 1983.

[5] Wuensche, A., and M.J. Lesser. The Global Dynamics of Cellular Automata; An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata, Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, Reading, MA, 1992.

[6] Wuensche, A., Complexity in 1D cellular automata; Gliders, basins of attraction and the Z parameter, Santa Fe Institute Working Paper 94-04-025, 1994.

[7] Wuensche, A., Classifying cellular automata automatically; finding gliders, filtering, and relating space-time patterns, attractor basins, and the Z parameter, Complexity, Vol.4/no.3, 47–66, 1999.

[8] Wuensche, A., Encryption using Cellular Automata Chain-rules, Examples implemented in DDLab, unpublished, 2000.

[9] Wuensche, A., The DDLab Manual (for ddlabx24), section 16.7 Setting a chain rule, `www.ddlab.org`, 2001.

[10] Wuensche, A., Encryption using Cellular Automata Chain-rules, unpublished, 2004.

[11] Wuensche, A.,Discrete Dynamics Lab: Tools for investigating cellular automata and discrete dynamical networks, updated for multi-value, Section 23 Chain rules and encryption, in Artificial Life Models in Software, eds. A.Adamatzky and M.Komosinski, chapter 11, 263-297, Springer. 2005

[12] Wuensche, A., Discrete Dynamics Lab (DDLab), software for investigating discrete dynamical networks, `www.ddlab.org` (1994-2006)

.

A cryptographic model based on the pre-image calculus of cellular automata

Gina M. B. Oliveira[1], Heverton B. Macêdo[1], Augusto A. B. Branquinho[1] and Marcelo J. L. Lima[2]

[1] Faculdade de Ciência da Computação,Universidade Federal de Uberlândia - UFU
Av. João Naves de Ávila, 2121- Campus Santa Mônica, Bloco B, sala 1B60
CEP: 38400-902 Uberlândia, MG.
heverton@pos.facom.ufu.br,gina@facom.ufu.br

[2] Programa de Pós-Graduação em Engenharia Elétrica, Universidade Presbiteriana Mackenzie

Abstract. This paper is about the application of cellular automata (CA) in cryptography. In the approach investigated here, the ciphering is accomplished by pre-image calculus while deciphering is performed by CA temporal evolution. We investigated the application in a cryptographic model of the pre-image calculus algorithm proposed by Wuensche [14] known as reverse algorithm. The viability of this algorithm when applied to any arbitrary plaintext is based on the prerogative that all configurations of CA lattices have at least one pre-image. We speculate that transition rules with chaotic dynamical behavior are more probable to exhibit this characteristic. Therefore, we investigate if it is possible to find rule sets that guaranty the existence of one pre-image for all possible CA lattices. Theses rule sets were found by using a genetic algorithm (GA) which was guided by the forecast dynamical behavior parameter named as Z. The results of our experiments show that, beyond the dynamics forecast performed by Z, other two characteristics are important: the symmetric level (S) and the balance between two components of Z named Z_{left} and Z_{right}.

1 Introduction

Several researchers investigated how CA can be employed as cryptography algorithms [12, 7, 2, 10, 9]. A special kind of CA rule is used as the secret key in Gutowitz's cryptographic model [2]. They are toggle transition rules. By this method, the plaintext is the initial lattice and P pre-images are successively calculated, obtaining the ciphertext. Starting from the ciphertext the receptor just needs to apply forward the transition rule by P times and the final lattice will correspond to the plaintext. An important point related to this model is that the lattice length increases of some cells for each pre-image processed, turning the ciphertext larger than the plaintext.

Some CA classification schemes have been used in the literature; for instance, Wolfram proposed a qualitative behavior classification in four dynamical classes, which is widely known [11]. The dynamics of a cellular automaton is associated to its transition rule. It has already been proved that the decision problem associated with forecasting CA dynamics is undecidable [1]. In order to help forecast CA dynamical behavior, several parameters have been proposed, directly calculated from the transition rule [3, 13, 8] Z parameter [13] is one of them.

The pre-image calculus proposed by [14] named reverse algorithm is investigated as a new cipher method. The advantages of this method are: (i) it can be applied to any kind of CA transition rule (not only toggle rules); (ii) the pre-image has the same length of the original lattice. However, the viability of this algorithm as a cipher method depends on the premise that any lattice has at least one pre-image. Besides, a desirable characteristic in any CA-based cipher method is that any perturbation in the initial lattice, which corresponds to the plaintext, propagates very fast along the cells to turn the original text unrecognizable from the ciphertext. Chaotic rules are more suitable to this task. Therefore, we performed an investigation about the usage of Z parameter to specify CA transition rules able to find pre-image for any arbitrary lattice.

Considering the application of the original reverse algorithm which does not have the disadvantage to be an increasing-length method, we have obtained important conclusions: the join specification of the requisites (i) Z, (ii) $\text{Z}_{left}/\text{Z}_{right}$ balance and (iii) symmetry (S) was shown to be a successful approach to define the transition rules to be used as secret keys. A final specification of these rules are Z_{righ} =1; 0.25 < Z_{left} < 0.5 and 0.25 < S < 0.5 or Z_{left} =1; 0.25 < Z_{righ} < 0.5 and 0.25 < S < 0.5. Experiments on Sec. 6 show that rules with this specification have high probability to find at least one pre-image for any arbitrary pre-image. Besides, using a spatial entropy measure it was possible to verify that rules with this specification exhibit a chaotic-like behavior. However, using the original reverse algorithm even using rules that satisfying all these requisites it is possible to fail when calculating a pre-image of some lattices. Therefore, to effectively use this method as a ciphering process, we need to adopt some kind of contour procedure. Here we propose to alternate the original and a modified reverse algorithm in the cipher process, using the second only when the pre-image calculus fails. This modified reverse algorithm adds extra bits on the pre-image, turning it larger than the original lattice. Sec. 7 discusses this new proposal.

2 Dynamical behavior and the Z parameter

The paper [11] proposed a qualitative behaviour classification of CA, which is widely known. It divides the CA rule space into four classes, according to the results of evolving the system from an arbitrary initial state: (1) homogeneous; (2) stable or periodic; (3) chaotic; (4) complex. Chaotic rules are sensible to perturbations on the initial configuration of the lattice and exhibit a high level of randomness in their temporal evolution. For this, they have been applied to several cryptographic systems [2, 9, 10].

The dynamics of a cellular automaton is associated with its transition rule. However, the decision problem associated with precisely forecasting the dynamical behavior of a generic cellular automaton, from arbitrary initial configurations, has been proved to be undecidable [1]. In order to help forecast the dynamic behavior of CA, several parameters have been proposed, directly calculated from the transition table. The basic idea associated to such kind of parameter is to perform a simple calculus over the output bits of the transition rule. It is expected that the result of this computation can estimate the CA behavior when the rule transition is applied by many time steps starting from any arbitrary lattice. Several parameters have been proposed in the literature [3, 8]; Z parameter is one of them [13]. The definition of Z parameter derived from the pre-image calculus algorithm. Z is a measure of the pre-imaging degree imposed by the automaton's rule, that is, it indicates if the number of possible pre-images is high (low Z) or low (high Z), for any arbitrary lattice configuration. Due to it, it is expect that when using CA rules with high Z there is a high probability to have at least one pre-image for the majority of possible lattices.

Z parameter is composed by Z_{left} and Z_{right}. Let us assume that part of a pre-image of an arbitrary configuration is known and that we want to infer the missing cell states, successively, from left to right. Z_{left} is defined as the probability that the next cell to the right in the partial pre-image has a unique value, and it is directly calculated from the transition table, by counting the deterministic neighborhoods. $Z_{right}$is the converse, from right to left. Z parameter is the greater of Z_{left} and Z_{right}. A detailed explanation about this parameter is given in [13]. In [8] an analysis of Z parameter is presented and the main conclusions are: (i) it is good discriminators of the chaotic behavior (ii) rules with a Z value close to 1 have high probability to be chaotic.

3 The reverse algorithm

An algorithm for CA pre-images calculation was proposed by [14]. This algorithm is known as reverse algorithm and it is based on the idea of partial neighborhoods. There are three different kinds of partial neighborhood: deterministic, forbidden and ambiguous. Figure 1 shows the three possibilities of neighborhood that can happen, when the calculation of the pre-image is being carried through from the left to the right. It is assumed that all the left $m-1$ cells of a neighborhood of m bits have already been calculated and the bit in boldface corresponds to the next bit (the rightmost cell) to be defined by the transition rule. Figure 1a shows an example of deterministic neighborhood (00**?**→0): there is only one neighborhood in the transition rule (00**0**→0) corresponding to the known cells of the partial neighborhood (00) in the pre-image and to the output bit in the original lattice (0). Figure 1b shows an example of ambiguous neighborhood (01**?**→0): for the partial neighborhood (01) there are two possible correspondent transitions in the rule (01**0**→0 and 01**1**→0) compatible with the output bit (0). In this case, the next bit can be either 0 or 1. Figure 1c shows an example in which it is not possible to determine the next bit for the partial pre-image (01**?**→1). Considering the partial neighborhood (01) there is no possible correspondent transitions in the rule compatible with the output bit (1). Thus, the third one is a forbidden partial neighborhood.

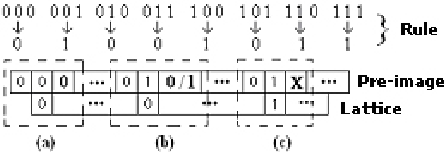


Fig. 1. Partial neighborhood: (a) deterministic (b) ambiguous (c) forbidden.

Before starting the calculus, r cells are added to each side of the lattice corresponding to the pre-image, where r is the CA radius. The algorithm starts the calculus of the pre-image, randomly initializing the $2{\times}r$ leftmost cells. Thus, the first partial neighborhood is formed; it can be deterministic, ambiguous or forbidden. If it is deterministic, the next bit is determined and the process continues. If the partial neighborhood is ambiguous, the next bit is randomly chosen, the other complementary value is stored in a stack and the calculus continues. If the partial neighborhood is forbidden, the next bit is impossible to obtain. In this case the calculus returns to randomly choose another initialization of the $2{\times}r$ leftmost cells and the process starts again. Whenever a next rightmost bit is defined, a new partial neighborhood is formed. The process to determine the next cell is the same as to determine the first one: (i) if the partial neighborhood is deterministic, the next bit is deterministically determined; (ii) if it is ambiguous a choice is made and its complementary is stored for future use; (iii) if it is forbidden the next bit is impossible and the process needs to return to the last choice stored in the stack. In the last situation, if all the values had been used and the stack is empty, the process stops because no pre-image is possible to obtain. This process continues until only the last $2{\times}r$ rightmost cells remains undetermined. If the CA boundary condition is periodic, these last cells need to be validated. The pre-image calculus is concluded verifying if the initial bits can be equal to the final $2{\times}r$ rightmost ones. If so, the extra bits added to each neighborhood side are discarded returning the pre-image to the same size of the original lattice. If no, the process returns again to use the last option stored in the stack.

The method proposed by [14] finds all the possible pre-images for an arbitrary lattice: if it is necessary to find all of them, the calculation must continue until all the ambiguities and values for the initial bits have been evaluated.

4 Toggle rules and Gutowitz's model

A CA transition rule is said to be a toggle one if it is sensible in respect to a specific neighborhood cell, that is, if any modification of the state on this cell necessarily provokes a modification on the new state of the central cell, considering all possible rule neighborhoods. Considering one-dimensional radius-1 CA, the neighborhood is formed by three cells and the state $a_i^{(t+1)}$ of the cell i in time $t+1$ is determined by the transition rule τ as $a_i^{(t+1)} = \tau\left(a_{i-1}^{(t)}, a_i^{(t)}, a_{i+1}^{(t)}\right)$. The rules can be sensible in respect to the left cell ($\tau[a_{i-1}^{(t)}, a_i^{(t)}, a_{i+1}^{(t)}] \neq \tau[a'^{(t)}_{i-1}, a'^{(t)}_{i}, a'^{(t)}_{i+1}]$, $\forall \left(a_{i-1}^{(t)} \neq a'^{(t)}_{i-1}\right)$),to the right cell ($\tau[a_{i-1}^{(t)}, a_i^{(t)}, a_{i+1}^{(t)}] \neq \tau\ [a'^{(t)}_{i-1}, a'^{(t)}_{i}, a'^{(t)}_{i+1}]$, $\forall \left(a_{i+1}^{(t)} \neq a'^{(t)}_{i+1}\right)$) and to the central cell ($\tau\ [a_{i-1}^{(t)}, a_i^{(t)}, a_{i+1}^{(t)}] \neq \tau[a'^{(t)}_{i-1}, a'^{(t)}_{i}, a'^{(t)}_{i+1}]$, $\forall \left(a_i^{(t)} \neq a'^{(t)}_{i}\right)$). For example, Figure 2 presents CA toggle rules: (a) right-toggle rule (b) left-toggle rule. The CA rule presented in Figure 2(c) is simultaneously sensible to the left and to the right cells (bidirectional toggle rule).

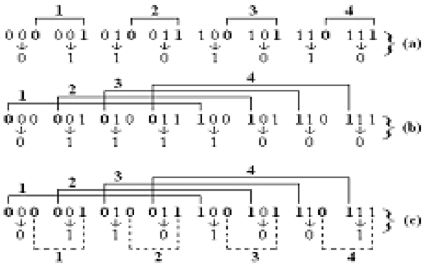


Fig. 2. Toggle rules (a) right , (b) left (c) right and left.

The paper [2] used irreversible CA with toggle rules — either to the right or to the left — for encrypting. Toggle rules turn possible to calculate a pre-image of any lattice starting from a random partial pre-image. Consider the initial lattice and the transition rule presented in Fig. 3. Two initial bits are randomly chosen to start the pre-image calculation and they are positioned in the extreme border of the lattice that is the opposite of the sensibility direction. In Fig. 3, the two initial bits are the rightmost cells because the transition rule is left-toggle. For example, it can be initialized with 01. The state of the next left cell is 0 because the only two possible transitions are 001→1 and 101→0. Once the state of the third cell is determined, the next step is to determine the state of the forth cell and the other cells successively up to completing the entire pre-image. The left-toggle property of the rule transition guarantees that all the pre-image cells can be obtained, step-by-step, in a deterministic way. The same process can be done for right-toggle rules and in this case the initial cells must be positioned in the leftmost side.

A toggle transition rule τ is used as the secret key in Gutowitz's cryptographic model [2]. The plaintext is the initial lattice and P pre-images are successively calculated, starting with random initial bits ($2\times r$ cells) at each step, where $r$is the radius. The ciphertext is given by the last pre-image obtained after P steps.

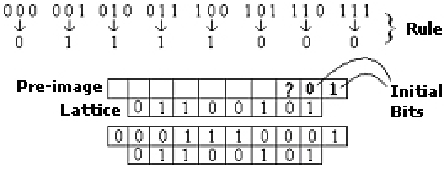


Fig. 3. Pre-image calculus using a left-toggle rule.

The decryption process is based on the fact that the receptor agent knows both τ and P. By starting from the ciphertext the receptor just needs to apply the transition rule τ forward by P steps and the final lattice will be the plaintext.

A fundamental characteristic in Gutowitz's model to guarantee pre-image for any lattice is the usage of a non periodic boundary condition. Thus, the final bits do not need to be validated in this method and they are calculated in the same manner as the others. When the pre-image is obtained the extra bits are not discarded and they are incorporated to the lattice. Therefore, it is always possible to obtain a pre-image for any initial choice of bits. As the length of initial bits is $2{\times}r$, it is possible to obtain different pre-images for each lattice. Therefore, the cellular automaton is irreversible. Let P be the number of pre-image steps, N the length of the original lattice and r the CA radius. The method adds $2{\times}r$ bits to each pre-image calculated and the size of the final lattice is given by $N + (2{\times}r{\times}P)$. For example, if 10 pre-images were calculated using a radius-1 rule starting from a plaintext of 16 bits, the size of the ciphertext would be 36.

A high degree of similarity between ciphertexts when the plaintext is submitted to a little perturbation was identified as a flaw in Gutowitz's model. The original model was altered by using bidirectional toggle CA rules (Fig. 2c) instead of one-directional ones in [9]. The experiments with this model show that the similarity flaw was solved with such a modification and it is protected against differential cryptanalysis [9]. However, the ciphertext increase in relation to the plaintext length remains in this model.

5 A new approach

The main goal of our work is to evaluate if the reverse algorithm proposed by [14] could be used as a cipher method. We started from the knowledge that as higher is the value of Z parameter calculated for a CA transition rule: (i) as higher is the probability of the cellular automation exhibits a chaotic behavior and (ii) as higher is the probability of a pre-image exists for any lattice. Then, the set of CA rules with Z equal to 1 was chosen as potential rules to use as secret keys in a cryptographic model based on the reverse algorithm.

In the experiments reported here, a genetic algorithm was used to find radius-2 and radius-3 CA rules having a specific characteristic. Z parameter was the main criterion applied to the fitness function to guide GA in the search for a set of rules.

The initial population is formed by 150 CA rules randomly generated. Each individual is a binary string with its length depending on the CA radius used: 32 bits for radius 2 and 128 bits for radius 3. The fitness function applied to each population's rule returns a value between 0 and 100. How much bigger is its fitness, closer is the rule to present the desired characteristic. For example, if it is desired that the rule has Z equal to 1, a rule with Z equal to 0.56 would have an evaluation equal to 56. Roulette wheel was used to select crossover pairs. A standard point crossover was used and a crossover rate of 60%. The mutation rate was of 8% per individual. The offspring is compared with the parents and the best ones survive for the next generation. This process is repeated 100 times, getting 150 rules showing the desired characteristic in the final population.

Each rule set generated by GA was employed in two different approaches using pre-image calculus. The first one refers to the original reverse algorithm proposed by [14], which uses periodic boundary condition, the final bits must be validated and the extra bits are discarded at each pre-image found. We call it as *pre-image calculus without extra bits*. The second approach is a variation of the original method in which the boundary condition is non periodic, the final bits are not validate and the extra bits are kept at each pre-image found, similar to Gutowitz's method described in Sec. 3. We call this method as *pre-image calculus with extra bits*.

6 Experiments

6.1 Initial experiments

It was realized experiments using CA with radius 1, 2 and 3. Considering radius-1 CA rules, all chaotic rules with $Z = 1$ were analyzed; they are 20 from the 256 elementary rules. Considering radius-2 and radius-3 CA rules, samples generated by GA with 150 transition rules with $Z = 1$ for each radius size were evaluated. All these rule sets (radius 1, 2 and 3) was applied to both pre-image methods explained in the last section. In all experiments each rule was tested verifying if it was able to calculate 10 consecutive pre-images starting from each lattice from a sample of 100 lattices with a fixed size. Each lattice sample was randomly generated. In this paper we show the results obtained using samples of lattices with 512 bits, except the last results in this section obtained using lattices of size 64.

The twenty radius-1 rules with chaotic behavior and Z=1 were first evaluated. For each rule evaluated, the percentage of lattices (out of 100) for which it was possible to calculate 10 consecutive pre-images was stored. This result was obtained for the both pre-image calculus method used. Considering pre-image calculus without extra bits, fourteen rules (out of 20) was able to obtain 10 consecutive pre-images starting from all the 100 lattices analyzed with 512 bits. On the other hand, six rules exhibit a converse behavior and could not finish the consecutive pre-images calculus for none of the lattices. Considering pre-image calculus with extra bits, all the twenty radius-1 rules was able to find 10 consecutive pre-images for all the 100 tested lattices.

The results obtained with radius-2 and radius-3 CA rules using pre-image calculus without extra bits are graphically presented in Fig. 4a. The radius-2 CA rules outperform the radius-3. However, nor all the radius-2 rules had returned 100% of success when calculating the 10 consecutive pre-images starting from all the lattices in the sample. Approximately 10 out 150 tested rules have failed starting from some lattices. Considering the radius-3 CA rules the results are worst: only 20 out 150 rules had returned 100% of success and approximately 80 of these rules have returned above 90% of performance.

On the other hand, Fig. 4b shows the results when the method with extra bits was applied to the same radius-2 and radius-3 CA rules. All these rules have returned 100% of success when calculating the 10 consecutive pre-images. It is important to say that this result was not possible to obtain using rules with Z $\neq$ 1. Therefore, it was possible to evidence that the pre-image calculus without extra bits, a variation of the original Wuensche and Lesser's reverse algorithm can be applied in a new cryptographic model with $Z = 1$ CA rules. In such method, the CA rules with $Z = 1$ are used as secret keys and the pre-image calculus algorithm are employed as the cipher method. The pre-image calculus are applied starting from the plaintext — used as the initial lattice — by P times, where P is a predefined value known by either sender and receiver. The final pre-image is transmitted to the receiver which has only to apply forward the same rule (secret key) by P times to recover the plaintext.

It is important to notice that transition rules with $Z = 1$ are more general than leftmost toggle rules or rightmost toggle rules employed in the methods discussed in [2, 9]. All these toggle rules (rightmost, leftmost or both) have Z=1, but the converse is not true. For example, a toggle rule sensible to the central cell has Z=1 but can not be applied in the previous models. Therefore, the key space for this new model, based on the variation of the reverse algorithm and using rules with $Z = 1$, will have a larger cardinality key space when compared to the previous methods, for the same specified radius. However, a cryptographic model based on the variation of the reverse algorithm employing extra bits will suffer the same disadvantage of the others: the ciphertext is larger than the plaintext. So, it is still an increasing-length cryptographic model and this increase is defined by *2PR,* where R is CA radius and P is the number of pre-image steps.

6.2 Symmetry and $\mathbf{Z}_{left}/\mathbf{Z}_{right}$ balance analysis

Due to the last arguments, we return our attention to the experiments using the original reverse algorithm trying to identify why in all analyzed radius it was possible to identify rules with 100% of success but there are other rules with null performance. Inspecting with well-taken care the null performance rules with radius 1 (8 bits of length) it was possible to observe an unusual characteristic in all of them: they are totally symmetric, that is, the first bit is identical to the last, the second is equal to the penultimate and so on (Ex: 01011010). Once noticed

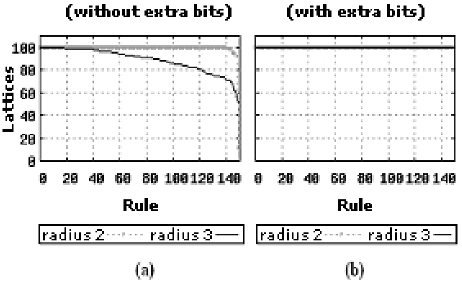


Fig. 4. Experiments using radius-2 and radius-3 CA rules with Z=1 (a) pre-image calculus without extra bits (b) pre-image calculus with extra bits.

this fact, we started to analyze the symmetry of radius-2 and radius-3 CA rules (32 and 128 bits of length), trying to relate this characteristic to the rule performance. It was possible to observe that the majority of low performance rules have higher levels of symmetry; even so they were not necessarily symmetrical in all the bits. Thus, we defined the symmetric level S as a new parameter of the rule that is important to specify the secret keys, besides Z parameter.

Other characteristics of the low performance rules with radius-2 and radius-3 were analyzed and one of them asked our attention: in the majority of low performance rules the two components of Z, named Z_{left} and Z_{right}, was close or equal to 1. Due to its proper definition considering a rule with Z equal to 1, either Z_{left} or Z_{right} necessarily must be equal the 1. However, in some rules the two components had high values: one equal to 1 and the other close or equal to 1. On the other hand, rules with good performance on pre-image calculus have only one of the components equal to 1. Thus, the unbalancing of the components Z_{left}/Z_{right} has also showed significant in the rule specification.

Aiming to confirm the relevance of symmetry level and Z_{left}/Z_{right} balance in the specification of the secret keys, the GA was executed again to find two samples of rules with different specifications, for each radius size. In the first one, the fitness evaluation was directly proportional to Z and to symmetry. It was possible to find 150 rules with $Z = 1$ and $S = 1$ for radius-2 and radius-3 CA. In the second one, the fitness evaluation was directly proportional to Z_{left} and Z_{right}. As a consequence, 150 CA rules with $Z_{left} = 1$ and $Z_{right} = 1$ (consequently $Z = 1$) was found for each radius size.

Two experiments were performed in which we try to calculate 10 consecutive pre-images using the original reverse algorithm and the two last samples of rules. The results of theses new experiments using radius-3 CA rules are presented graphically in Fig. 5, in which the performance of $Z = 1$ and $S = 1$ rules sample and $Z_{left} = 1$ and $Z_{right} = 1$ rules sample are shown. The curve related to the experiment using radius-3 and $Z = 1$ rules (Fig. 4a) was replicated to facilitate the comparison. It is possible to observe that the performance of the rules in the new samples fell drastically, confirming that these characteristics also must

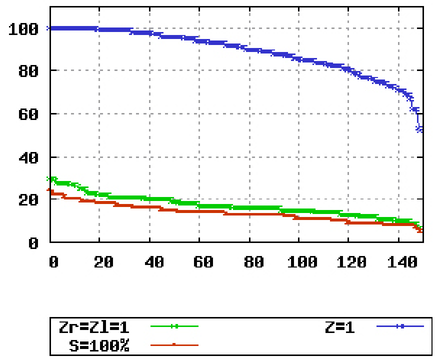


Fig. 5. Experiments using radius-3 CA and the pre-image calculus without extra bits. Rules generated with: (a) Z=1 (b) Z=1 and S=100% (c) Z_{left}=1 and Z_{right}=1.

be prevented. In other words, the rules to be good secret keys — considering its ability to have at least one pre-image — must not only have Z equal to 1 but they must not have a high level of symmetry and must not have a high Z_{left}/Z_{right} balancing.

GA was modified to find a sample of rules with the following characteristics: $S < 0.25$, $Z = 1$ and $Z_{left} < 0.25$ (consequently, $Z_{right} = 1$). A significant improvement happened in the performance of the rules in relation to their capacity to calculate 10 consecutive pre-images. In the experiment using rules searched with only the specification $Z = 1$, approximately 20 out of 150 rules returned 100%. In the experiment using the specification $S < 0.25$, $Z_{right} = 1$ and $Z_{left} \quad < 0.25$ all the 150 rules returned 100% of performance. However, when we carefully analyze the pattern diagrams generated by these rules we discover that the rules generated with this specification have a very simple behavior: they only provoke a kind of shift in the lattice. Therefore, even that these $Z = 1$ rules have a good performance to calculate the pre-images, they can not be used as secret keys because they are not chaotic rules, they are fixed-point.

6.3 Final experiments

We have evidenced that rules with high level of symmetry and high Z_{left}/Z_{right} balancing are not good to use in the generic pre-image calculus. So, we performed a final experiment using rules with $0.25 < S < 0.5$ and $0.25 < Z_{left} < 0.5$ and $Z_{right} = 1$ (consequently, $Z = 1$). GA was modified to find rules with these characteristics and the performance of 100 radius-2 CA rules was evaluated calculating 128 consecutive pre-images starting from 1000 random lattices of 512 bits. All the rules were able to calculate the 128 consecutive pre-images for all the 1000 512-bits lattices.

Aiming to evaluate the randomness imposed by this consecutive calculus as a cipher process, we calculate the spatial entropy related to a perturbation in the initial lattice. This evaluation was performed as follows: for each one of the 1000 tested lattices we generated a similar lattice which differs by the original in only 1 of the 512 bits; that is, we performed a simple perturbation in each 512-bits lattice X obtaining a correspondent similar lattice X'. Each related pair of initial lattices (X, X') was submitted to the pre-image calculus obtaining a pair of ciphertexts (Y, Y') and XOR operation was applied using Y and Y' resulting in a string D with 512 bits, which represents the difference between Y and Y'. A spatial entropy calculus was performed in each D related to each initial lattice X of the sample of 1000 512-bits lattices. Figure 6 presents the mean entropy found for each radius-2 rule evaluated (calculated over 1000 lattices).

Table 1. Entropy of some radius-2 rules evaluated

Rule	E_{mean}	E_{min}	E_{max}
10100110011001101010101010011010	0.901	0.883	0.930
10101001010110011010100110100110	0.908	0.886	0.928
10010101010110100101010101010110	0.889	0.424	0.928
01010110010101101001011001010101	0.876	0.385	0.929
01100110011010010110100101101010	0.909	0.885	0.928

The average of the mean entropy obtained for all the 50 evaluated radius-2 rules is 0.8857. As we can see in the figure the lower mean entropy found was 0.75 and the higher mean entropy was around 0.91. A spatial-entropy above 0.7 is enough to guarantee the randomness of a binary string. Therefore, we can conclude that this last sample of radius-2 CA rules has a good performance in the pre-image calculus and they exhibit chaotic-like behaviors. Table 1 shows 5 of these rules; its binary code is presented (from 00000 to 11111), the mean entropy (E_{mean}), the minimum entropy (E_{min}) and the maximum entropy (E_{max}) found in the tests with the sample of 1000 lattices. All of them were able to calculate the 128 consecutive pre-images for all the 1000 512-bits lattices.

Aiming to compare these results with the rules with fixed-point behavior this evaluation was also performed for the sample generated with $S < 0.25$, $Z = 1$ and $Z_{left} < 0.25$: the average of the mean entropy obtained for all the 150 rules is 0.5604; the lower mean entropy found was 0.1152 and the higher mean entropy was around 0.7760. These low values of entropy corroborate our visual identification of the fixed-point behavior.

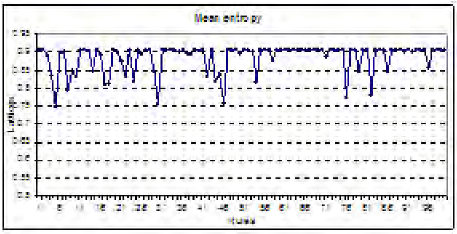


Fig. 6. Mean entropy of the difference obtained using 100 radius-2 CA rules submitted to 1000 pairs of similar initial lattices.

7 A proposal of a new cryptographic model with a variable-length ciphertext

Using all the characteristics investigated in last sections to generated secret keys it was possible to obtain rules with a high probability to find at least one pre-image for any lattice and a good perturbation spread. However, even the better rules evaluated can fail when the pre-image calculus is applied to some lattice. Thus, one alternative to propose a method based on the original reverse algorithm is to adopt a contour procedure to apply when the pre-image calculus fail. As the secret key specification previous discussed gives a low probability to this failure occurrence, we expect to rarely use this contour procedure but it guarantees the possibility to cipher any plaintext.

The contour procedure proposed here is to add extra bits only when the pre-image is not possible to calculate. The cipher process is defined by the calculus of P consecutive pre-images starting from a lattice of size N corresponding to the plaintext. The secret key is a radius-R CA rule generated with the specification: Z_{right} =1; 0.25 $< Z_{left} <$ 0.5 and 0.25 $< S <$ 0.5 or Z_{left} =1; 0.25 $< Z_{right} <$ 0.5 and 0.25 $< S <$ 0.5. Suppose that the algorithm started to calculate the pre-images using the reverse algorithm and it fails in the K-th pre-image such that $K \leq P$.In such situation the cipher process uses the modified reverse algorithm with extra bits to calculate the K-th pre-image. Thus, the K-th pre-image will have $N + 2R$ cells. The cipher process return again using the original reverse algorithm (without extra bits) to calculate the remaining pre-images. If all the subsequent pre-images calculus succeeds the final ciphertext will have a size of $N + 2R$.If the pre-image calculus fails again, the cipher process changes to the modified reverse algorithm and add more $2R$ bits. If the process fails in F pre-images ($F \leq P$)the final lattice will have *N+2FR.* Therefore, in theory, starting from a lattice of N cells, the size of the ciphertext after P pre-images calculus can be:

$$N \leq \text{ciphertext size} \leq N + 2PR \tag{1}$$

For example, if N= 512, P= 32 and$R$= 5 the size of the ciphertext will be situated between 512 and 832 bits. However, we expected that in practice the ciphertext size will be next to 512 due to the characteristics of the rule used as secrete key. It is important to note that using the same parameters the ciphertext obtained using Gutowitz's model will have exactly 832 bits — the worst and improbable situation in the new proposed method. Thus, it is a variable-length cryptographic model.

Another aspect that must be addressed in this proposal is related to the information needed by the receptor to do the decipher process. Based only on the size of the ciphertext transmitted, the receptor can obtain the number of pre-images in which the reverse algorithm had failed. For example, for the same parameters used in the previous example, if the ciphertext has size 532, the number of fails had been two ($F = 2$), since $512+2\times2\times5=532$.However,this information is not enough to recover the cipher process. The receptor need to know not only the number of fails but exactly in which pre-images the reverse algorithm had failed and the extra bits was added. In a fail occurrence, we propose to transmit a fixed binary block with size equal to the number of pre-images calculated(P)such that the first bit is related to the first pre-image calculated, the second bit to the second pre-image and so on. If the K-th pre-image fail in the original reverse algorithm, the K-th bit has value 1; otherwise it has value 0. Suppose that the 2 fails related to the previous example had happened in the calculus of the 5-th and 14-th pre-images. So, the fixed block transmitted is

0000100000000100.

The case that no fail occurred does not need to transmit the fixed block. Using this approach the amount of bits actually transmitted is given by:

$$N \leq \text{bits transmitted} \leq N + P + 2PR \tag{2}$$

For example, if $N = 512$, $P = 32$ and $R = 5$ the size of the ciphertext will be situated between 512 ($F = 0$) and 848 ($F = 32$) bits in theory and next to 512 in practice.

8 Additional remarks

The results presented in this work derived from several studies carried through since 2000 in Brazil under the coordination of the first author of this work. The initial motivation of these studies drifts of the analysis of the cryptographic model proposed by Gutowitz. By this analysis, it was evidenced that this method provokes a strong increase of the ciphertext in relation to the plaintext, discouraging its practical use, especially in respect to the transmission of the ciphered data. In the first study performed [4], it was investigated the use of radius-1 CA rules with Z=1 with the reverse algorithm in a cipher method. From this study, it was clearly that the *Z equal to 1* condition of a rule would not be enough to employ it as a cryptographic key. In a subsequent study [5], the symmetry and Z_{left} /Z_{right} balance had shown promising in a more restricted specification of rules that could in fact be viable as keys. In a complementary study [6], it was clearly that in the search for rules with 100% of pre-image guarantee, we could be in approaching to rules with a fixed-point behavior and moving away from rules with the desired chaotic behavior. More recently, with the aid of a spatial entropy measure and employing a subgroup of Z=1 rules specified using adequate symmetry level and Z_{left} /Z_{right} balancing, it was able to verify that these rules have high probability of pre-image for any lattice and exhibits chaotic behavior. These rules are characterized by $Z_{left} = 1$; $0.25 < Z_{right} < 0.5$ and $0.25 < S < 0.5$. From these last results and of all the previous analyzes, a proposal of a new cryptographic method was elaborated in this work.

We have just taken knowledge about two works written by Andrew Wuensche in which he had also considered the use of his reverse algorithm as a cryptographic method. The first reference is a book chapter about DDLab environment, which was elaborated by him for the study of cellular automata and discrete networks [15]. In this chapter, the idea to use the reverse algorithm is presented together with the employment of chain-rules as secret keys, that according to author, would tend to present only one pre-image for all possible lattice. Even so the concept and characterization of the chain-rules is not presented in this chapter. However, the chain rules had been defined previously in DDLab manual [17] in which it is said that they are suitable for encryption. In the second reference [16], not yet published, the idea is more elaborated and the "chain-rules" are characterized with the use of Z parameter and their components Z_{left} and Z_{right}: these rules must have Z equal to 1 but with Z_{left} different of Z_{right}; that is, the both components (Z_{left} and Z_{right}) can not be equal a 1. This rule characterization was reached by Wuensche based on his DDLab's studies about the Garden-of-Eden density related to basins of attraction. According to personal communications with the author, he had written his initial ideas of this paper in 2000 and it having remained not published for different reasons.

Although we have not done a detailed analysis of his work yet, we believe that they are similar but there are significant differences. We intend to compare his model with ours in a next future.

9 Conclusions

This work investigates the application of the reverse algorithm proposed by [14] as a cipher method in a cryptographic system. The usage of GA in the search of valid secret keys and Z parameter in their specification are also evaluated.

A variation of the reverse algorithm in which extra bits are added in each pre-image calculus has revealed viable of being applied in a cipher process, having been enough to use rules with Z equal to 1 as secret keys. A genetic algorithm can be easily used in the generation of rules of such kind. In relation to the previous methods proposed in [2] and [9], this method has the advantage to use more general rules. Consequently, it has a larger cardinality key space for the same CA radius. However, a disadvantage of the previous methods remains: a significant increase of the ciphertext in relation to the original plaintext.

Considering the application of the original reverse algorithm which does not have the disadvantage to be an increasing-length method, we have obtained important conclusions: the join specification of the requisites (i) Z, (ii) Z_{left}/Z_{right}

balance and (iii) symmetry (S) was shown to be a successful approach to define the transition rules to be used as secret keys. A final specification of these rules are $Z_{righ} = 1$; $0.25 < Z_{left} < 0.5$ and $0.25 < S < 0.5$ or $Z_{left} = 1$; $0.25 < Z_{righ} < 0.5$ and $0.25 < S < 0.5$. However, using the original reverse algorithm (without extra bits) even using rules that satisfying all these requisites it is possible to fail when calculating a pre-image of some lattices. Therefore, to effectively use this method as a ciphering process, we need to adopt some kind of contour procedure. Here we propose to alternate the original reverse algorithm and the modified reverse algorithm in the cipher process, using the second only when the pre-image calculus fails. This contour procedure needs to add some bits to the ciphertext in relation to the plaintext size when a fail occurs: (i) a fixed number of bits, equal to the number of pre-images P and (ii) a variable number of bits due to eventual fails, *2R bits* for each fail, where R is the CA radius. *We expected that in practice few fails happens and the ciphertext size will be equal or close to the plaintext.* We have implemented the proposed cipher method and we will submit it to more robust tests to confirm this expectation.

The approach discussed here uses a variable-length method to contour an eventual failure in the pre-image calculus. Another possible approach is trying to find a manner to guarantee that the cellular automaton will calculate at least one pre-image for any possible lattice. We are conducting a new investigation related to this way in which the problem is fixed using non-homogeneous CA.

10 Acknowledgements

GMBO thanks CNPq and FAPEMIG support. The authors thanks to A. Wuensche for valuable and critical comments.

References

[1] Culik II, K., Hurd, L. e Yu, S. (1990) Computation Theoretic Aspects of Cellular Automata. Physica D, 45:357-378

[2] Gutowitz, H. (1995) Cryptography with Dynamical Systems, In: Cellular Automata and Cooperative Phenomena, Kluwer Academic Press.

[3] Langton, C.G. (1990). Computation at the Edge of Chaos: Phase Transitions and Emergent Computation. Physica D, 42:12-37.

[4] Lima, M. J. L. e Sales, T. C. (2002). Cellular automata pre-image calculus in cryptography. Monograph of a scientific initiation research. Universidade Presbiteriana Mackenzie, Sao Paulo, Brazil. In Portuguese. (Original title in Portuguese: "C·lculo da Pre-Imagem em Automatos Celulares para Criptografia").

[5] Lima, M. J. L. (2005) Cryptography based on the generic cellular automata pre-image calculus. Master in Science Dissertation. Universidade Presbiteriana Mackenzie, Sao Paulo, Brazil. In Portuguese. (Original title in Portuguese: "Criptografia Baseada no Calculo Generico de Pre-Imagens de Automatos Celulares").

[6] Macedo, H (2007). A new cryptographic method based on the pre-image calculus of chaotic, non-homogeneous and non-additive cellular automata. Master in Science Dissertation. Universidade Federal de Uberlandia, Uberlandia, Brazil. In

Portuguese. (Original title in Portuguese: "Um novo metodo criptografico baseado no calculo de pre-imagens de automatos celulares caoticos, nao-homogeneos e nao-aditivos").

[7] Nandi, S., Kar, B., and Chaudhuri, P. (1994) Theory and Applications of CA Automata in Cryp-tography, IEEE Trans. Comp.,43:1346-1357.

[8] Oliveira, G.M.B., de Oliveira, P. e Omar, N. (2001) Definition and applications of a five-parameter characterization of 1D cellular automata rule space. Artificial Life, 7(3):277-301,

[9] Oliveira, G.M.B., Coelho, A.R. e Monteiro, L.H.A. (2004) Cellular Automata Cryptographic Model Based on Bi-Directional Toggle Rules, Intern. Journal of Modern Physics C, 15:1061-1068.

[10] Sen, S., Shaw, C., Chowdhuri, R., Ganguly, N. and Chaudhuri, P. (2002) Cellular Automata Based Cryptosystem, Proc. Int'l Conf. Information and Comm. Security (ICICS02), 303-314.

[11] Wolfram, S. (1984) Universality and Complexity in Cellular Automata. Physica D, 10:1-35.

[12] Wolfram, S. (1986) Cryptography with cellular automata, Advances in Cryptology: Crypto '85 Proceedings, LNCS, 218:429-432.

[13] Wuensche, A. (1999) Classifying Cellular Automata Automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins and the Z parameter. Complexity, 4 (3):47-66.

[14] Wuensche, A., Lesser, M. (1992) Global Dynamics of Cellular Automata. Addison-Wesley.

[15] Wuensche, A. (2005) Discrete Dynamics Lab: tools for investigating cellular automaton discrete dynamical networks. In: A.Adamatzky and M.Komosinski, eds. Artificial Life Models in Software, chapter 11, 263-297.

[16] Wuensche, A. (2004) Encryption using Cellular Automata Chain-rules. Unpublished manuscript.

[17] Wuensche, A. (2001) DDLab Manual. Avaliable in `http://www.ddlab.org/#manual`.

A definition of conservation degree for one-dimensional cellular automata rules

Angelo Schranko[2], Paulo E. Florencio[1], Rogério M. Ywassa[1], and
Pedro P. B. de Oliveira[12]

[1] Universidade Presbiteriana Mackenzie – Faculdade
de Computação e Informática
[2] Universidade Presbiteriana Mackenzie – Pós-Graduação
em Engenharia Elétrica
Rua da Consolação 896, Consolação – 01302-907 São Paulo, SP – Brazil
quinternion@yahoo.com.br,peflorencio@gmail.com,rywassa@gmail.com,pedrob@
mackenzie.br

Abstract. Decidability of the number-conservation property of one-dimensional cellular automata rules can be established by necessary and suficient conditions given by Boccara and Fukś Nevertheless, a related open question would ask for a definition of the intermediate conservation degree of a rule. Based upon those conditions, a proposal is made herein of the intermediate conservation degree for one-dimensional cellular automata rules. Empirical measures of this quantity are also defined and then related results of computational experiments are presented. Possible relations between theoretical and experimental results are also investigated.

1 Introduction

Cellular automata (CAs) are a class of discrete dynamical system where homogeneous, local, short-range interactions between their components can result in emergent behaviour [1]. Due to their distributed nature and local processing, they can play important role in complex systems modeling [2]. Nagel and Schreckenberg [3] proposed a one-dimensional, 2-state CA based model of highway traffic flow where the number of cars remains constant during system evolution. The success of that model depends on the fact that the evolution rule satisfies a certain quantity conservation law, that is, the rule must be number-conserving [4]. Indeed, given a one-dimensional CA rule, it is always possible to decide whether or not the rule is number-conserving, by means of the Boccara-Fukś conditions [5]; nevertheless, nothing can be said about the intermediate conservation degree, since such a quantity has apparently not yet been defined in the literature. The purpose of this paper is to contribute in filling up this gap. Understanding the meaning of conservation degree for non-conservative CAs may be useful in open systems modeling like biological and thermodynamical systems [2].

In what follows, this section provides general background material on CAs and on Boccara-Fukś conditions [5]. The next section then introduces a definition of conservation degree for one-dimensional CA rules that is consistent for rules belonging to the same dynamical equivalence class. In Section 3, an empirical conservation degree is defined, which is based on the normalisation of the sum of the proportion of number-conserving configurations, and then main results of the analyses of its asymptotic behaviour for all elementary rules are presented. Comparative results between theoretical and experimental conservation degrees are also considered. In addition, a new empirical measure of conservation degree for one-dimensional CA rules is proposed, that suggests an alternative interpretation of conservation in the present context; also, attempts are made to relate the empirical and theoretical conservation degrees. Finally, Section 4 discusses the conclusions and possible extensions of this work.

1.1 Cellular automata

CAs are fully discretised dynamical systems, since their spatial composition, temporal evolution and state variables are all given in discrete domains. All the processing is distributed, so that computations occur locally, without a central processing unit or memory [2]. Despite their simplicity, CAs may produce complex patterns useful in modeling natural phenomena. Basically, a CA consists of an n-dimensional array of identical finite state machines (cells), where each cell takes on a state drawn from a discrete set of values. Its spatio-temporal dynamics is determined by a state transition function, that depends on the state of the neighbourhood cells at a given time t. One dimensional CAs may be defined as follows [5]. Let $S : \mathbb{Z} \times \mathbb{N} \rightarrow \mathcal{Q}$ a function that satisfies the equation:

$$s(i, t+1) = f(s(i - r_\ell, t), s(i - r_\ell + 1, t), \ldots, s(i, r_r, t)). \tag{1}$$

For all $i \in \mathbb{Z}, t \in \mathbb{N}$ where $\mathbb{Z}$ is the set of integers, $\mathbb{N}$ is the set of nonnegative integers and $\mathcal{Q} = \{0, 1, \ldots, q-1\}, q \in \mathbb{N}$, a finite set of states. The state of cell i at time t is given by $s(i, t)$, as represented in Fig. 1. A configuration is any element $S \in \mathcal{Q}^{\mathbb{Z}}$. In this work only finite CAs are considered, so that $\mathcal{Q}^{\mathbb{Z}}$ is replaced by $\mathcal{Q}^L, L \in \mathbb{Z}$ being the length of the configuration.

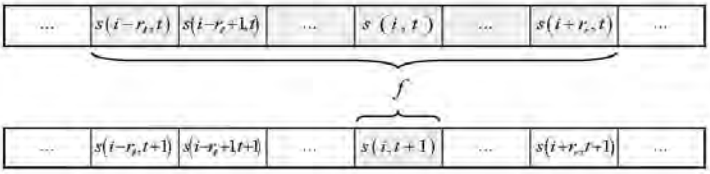


Fig. 1. Local evolution of a one-dimensional CA with neighborhood $n = r_\ell + r_r + 1$.

Let $f : \mathcal{Q}^n \to \mathcal{Q}$ be a local transition function (or rule), $n = r_\ell + r_r + 1$, where r_ℓ and $r_r \in \mathbb{Z}^+$ are, respectively, the left and right radii of the rule, referred to as an n-input rule. Following Wolfram's lexicographic ordering scheme [2], each transition function f can be assigned a rule number $N(f) \in \mathbb{Z}^+$ such that:

$$N(f) = \sum_{(x_1,x_2,\ldots,x_n)\in\mathcal{Q}^n} f(x_1, x_2, \ldots, x_n) q^{q^{n-1}x_1 + q^{n-2}x_2 + \cdots + q^0 x_n} \tag{2}$$

Since there are q^n distinct neighbourhoods, each of them mapping one of q distinct states, this entails a total of $\underbrace{q \ldots q \ldots q}_{q^n} = q^{q^n}$ distinct rules, which defines a rule space. In particular, when $n = 3$ and $q = 2$, a set with 256 rules defines the 'elementary' space [2]. But regardless of the space at issue, distinct rules may exhibit equivalent dynamical behaviours, when they relate to each other through certain particular transformations, as shown in expression (3). Accordingly, rule spaces can be partitioned into dynamical equivalence classes [2]. In the elementary space, 88 equivalence classes can be generated by applying the operators of conjugation and reflection, denoted respectively by C and R, and their composition CR (or, equivalently, RC), defined as follows [5]:

$$\begin{aligned} &Cf(x_1, x_2, \ldots, x_n) = q - 1 - f(q - 1 - x_1, q - 1 - x_2, \ldots, q - 1 - x_n) \\ &Rf(x_1, x_2, \ldots, x_n) = f(x_n, x_{n-1}, \ldots, x_1) \end{aligned} \tag{3}$$

1.2 One-dimensional CA rules

Consider the one-dimensional CA rule where $N(f) = 170$, $n = 3$ e $q = 2$. From (2) it follows that:

$$\begin{aligned} f(0,0,0) = 0, f(0,0,1) = 1, f(0,1,0) = 0, f(0,1,1) = 1 \\ f(1,0,0) = 0, f(1,0,1) = 1, f(1,1,0) = 0, f(1,1,1) = 1 \end{aligned} \tag{4}$$

It is easy to see that $\forall x_1, x_2, x_3 \in \mathcal{Q}$, $f(x_1, x_2, x_3) = x_3$. So, for any configuration with length $L \geq n$, according to [4], condition (5) should be satisfied:

$$\begin{aligned} f(s(1,t), s(2,t), s(3,t)) + f(s(2,t), s(3,t), s(4,t)) + \cdots \\ + f(s(L,t), s(1,t), s(2,t)) = s(1,t) + s(2,t) + \cdots + s(L,t) \end{aligned} \tag{5}$$

Thus, the sum of all states remains constant as the system is iterated. This is a consequence of the number-conserving indexnumber-conserving rule nature of rule 170, according to (5). Note that rule is equivalent to the application of the *shift-left* operator to the initial configuration, which is clearly number-conserving (as depicted in Fig. 2). Number conserving CA rules may be regarded as evolution operators where the number of interacting particles remains constant as the system evolves, caracterising an isolated system [5].

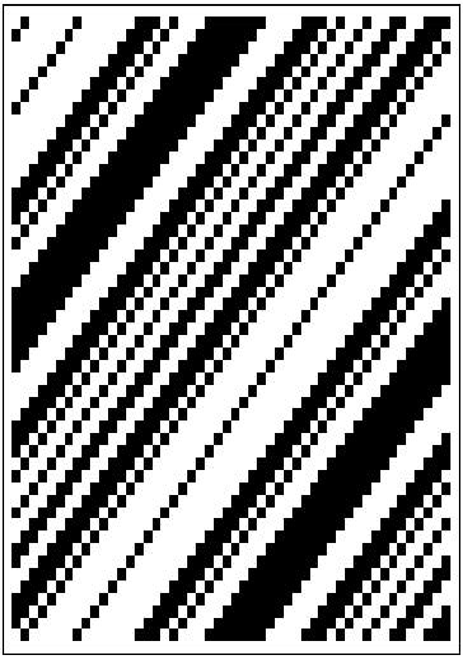


Fig. 2. Spatio-temporal diagram of elementary rule 170, from a random initial configuration of length 50, for 50 iterations. Quantity-conservation is clearly noticed during the evolution.

As proved by Boccara and Fukś [5], an n-input one-dimensional CA rule is number-conserving if, and only if, $\forall (x_1, x_2, \ldots, x_n) \in \mathcal{Q}^n$, the rule satisfies:

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) = x_1 + \sum_{k=1}^{n-1} (f(\underbrace{0, 0, \ldots, 0}_{k}, x_2, x_3, \cdots, x_{n-k+1}) - \\ f(\underbrace{0, 0, \ldots, 0}_{k}, x_1, x_2, \cdots, x_{n-k}) \end{aligned} \tag{6}$$

In particular, it is clear that, if f is number conserving, then:

$$f(0, 0, \ldots, 0) = 0 + \sum_{k=1}^{n-1} (f(\underbrace{0, 0, \ldots, 0}_{k}, 0, \ldots, 0) - f(\underbrace{0, 0, \ldots, 0}_{k+1}, 0, \ldots, 0)) = 0 \tag{7}$$

For instance, it follows from (6) that (4) refers to a number-conserving CA rule, since:

$$\begin{aligned}
&f(0,0,0) = 0 \Leftrightarrow 0 = 0 \rightarrow True\\
&f(0,0,1) = f(0,0,1) - f(0,0,0) \Leftrightarrow 1 = 1 - 0 \rightarrow True\\
&f(0,1,0) = f(0,1,0) - f(0,0,0) \Leftrightarrow 0 = 0 - 0 \rightarrow True\\
&f(0,1,1) = f(0,1,1) - f(0,0,0) \Leftrightarrow 1 = 1 - 0 \rightarrow True\\
&f(1,0,0) = 1 + 2f(0,0,0) - f(0,0,1) - f(0,1,0) \Leftrightarrow 0 = 1 + 2.0 - 1 - 0 \rightarrow True\\
&f(1,0,1) = 1 + f(0,0,0) - f(0,1,0) \Leftrightarrow 1 = 1 + 0 - 0 \rightarrow True\\
&f(1,1,0) = 1 + f(0,1,0) - f(0,1,1) \Leftrightarrow 0 = 1 + 0 - 1 \rightarrow True\\
&f(1,1,1) = 1 \Leftrightarrow 1 = 1 \rightarrow True
\end{aligned} \tag{8}$$

Now, since for all number-conserving rules $f(0,0,\ldots,0) = 0$, a further step can be taken from Boccara-Fukś conditions, for all neighbourhoods starting with $x1 = 0$. In order to realise that, it suffices to introduce the latter considerations into (6) and then simplify it, as follows:

$$\begin{aligned}
f(0,x_2,\ldots,x_n) = 0 + \sum_{k=1}^{n-1}(f(&\underbrace{0,0,\ldots,0}_{k},x_2,x_3,\cdots,x_{n-k+1}) -\\
&f(\underbrace{0,0,\ldots,0}_{k+1},x_2,\ldots,x_{n-k})) =\\
&f(0,x_2,\cdots,x_n) - f(0,0,\cdots,0) =\\
&f(0,x_2,\cdots,x_n) \rightarrow True
\end{aligned} \tag{9}$$

This follows from the fact that, for each k, the addition of the second term of the sum with the first term for $k+1$ is always zero; for instance, while for $k = 1$ the second term of the sum yields $f(0,0,x_2,\ldots,x_{n-1})$, the first term for $k+1$ is $-f(0,0,x_2,\ldots,x_n1-1)$. As a consequence, q^{n-1} tautologies become evident in (6), so that decidability of the number-conservation property can, in fact, be established by checking only $q^n - q^{n-1} + 1$ conditions, instead of the q^n conditions of the original Boccara-Fukś proposal. This provides a simplified algorithm for achieving the conservation decision, at a lower computational cost.

2 Conservation degree of one-dimensional CA rules: An analytical proposal

Since the number-conservation property is preserved for rules in the same equivalence class of dynamical behaviour [5], the conservation degree should also be. So, by applying conjugation and reflection operators to (6), the following conditions must be satisfied, $\forall(x_1, x_2, \ldots, x_n) \in \mathcal{Q}^n$:

$$\begin{gathered}Cf(x_1, x_2, \ldots, x_n) = x_1 + \sum_{k=1}^{n-1}(Cf(\underbrace{0, 0, \ldots, 0}_{k}, x_2, x_3, \ldots, x_{n-k+1}) - \\ Cf(\underbrace{0, 0, ..., 0}_{k}, x_1, x_2, ..., x_{n-k}))\end{gathered} \tag{10}$$

$$\begin{gathered}Rf(x_1, x_2, \ldots, x_n) = x_1 + \sum_{k=1}^{n-1}(Rf(\underbrace{0, 0, \ldots, 0}_{k}, x_2, x_3, \ldots, x_{n-k+1}) - \\ Rf(\underbrace{0, 0, \ldots, 0}_{k}, x_1, x_2, \ldots, x_{n-k}))\end{gathered} \tag{11}$$

$$\begin{gathered}RCf(x_1, x_2, \ldots, x_n) = x_1 + \sum_{k=1}^{n-1}(RCf(\underbrace{0, 0, \ldots, 0}_{k}, x_2, x_3, ..., x_{n-k+1}) - \\ RCf(\underbrace{0, 0, \ldots, 0}_{k}, x_1, x_2, \ldots, x_{n-k}))\end{gathered} \tag{12}$$

Let Φ, Φ_C, Φ_R and Φ_{CR} be the equation sets from (6), (10), (11) and (12), respectively, for $n \in \mathbb{N}$ fixed. Hence, the theoretical conservation degree of one-dimensional CA rules can be defined as follows (where $|C|$ means the cardinality of set C):

$$\gamma = \frac{|\{\varphi \in \Phi \bigcup \Phi_C \bigcup \Phi_R \bigcup \Phi_{CR} : \varphi \text{ is True}\}|}{|\Phi \bigcup \Phi_C \bigcup \Phi_R \bigcup \Phi_{CR}|} \tag{13}$$

Therefore, the maximum possible conservation degree value is set to $\gamma = 1$, in the case of a number-conserving CA rule, and the minimal value is $\gamma = 0$. In this way, given a one-dimensional CA rule, γ refers to the proportion of conservation equations that are satisfied, in relation to the total number of conservation equations.

In particular, for $n = 3$ and $q = 2$ (i.e., in the elementary rule space), a further simplification is straightforward. For that, one can easily see that (11) is equivalent to (6), as it suffices to apply the reflection operator in (11) and notice that $Rf(x_n, x_{n-1}, \ldots, x_1) = f(x_1, x_2, \ldots, x_n)$, which results in identical equations. Analogously, the same argument can be used to show that the set of equations in (10) and (12) are also equivalent, by applying the reflection operator in (12). As a consequence, (6) and (10) become necessary and sufficient to define a measure that is identical among rules of the same equivalence class, thus entailing (13) to be simplified as follows:

$$\gamma = \frac{|\{\varphi \in \Phi \bigcup \Phi_C : \varphi \text{ is True}\}|}{|\Phi \bigcup \Phi_C|} \tag{14}$$

As illustration, Fig. 3 shows the value of γ for all elementary rules. Notice, for instance, that 5 rules display $\gamma = 1.0$, which correspond to the number-conserving elementary rules 170, 184, 226 and 240; also, 8 rules appear with

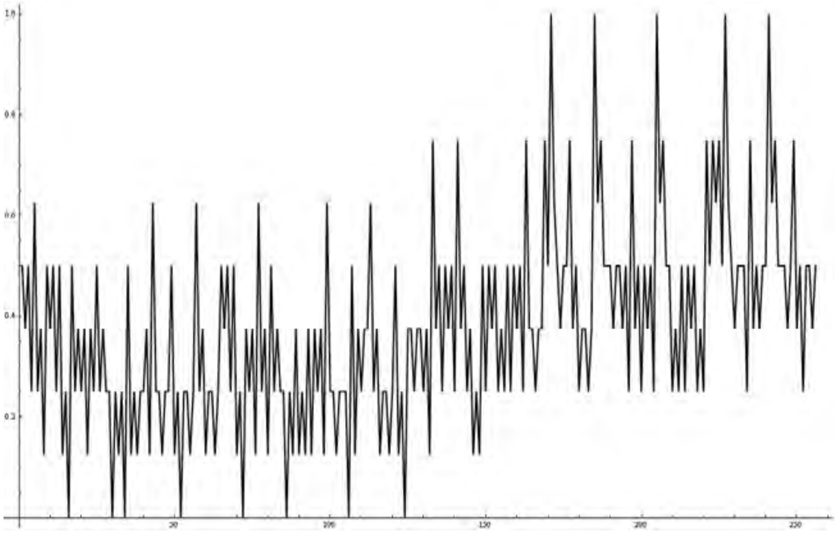


Fig. 3. Conservation degree γ for every elementary rule, in order of their number.

$\gamma = 0$, which are the rules 15, 29, 33, 51, 71, 85, 105 and 123. Since for q=2 and n=3 Boccara-Fukś conditions can be established by applying (5) to the configuration $(0, 0, x_1, x_2, x_3)$, it was observed that rules with $\gamma = 1.0$ satisfy (5) for all these configurations and for rules with $\gamma = 0$, no conditions are satisfied at all, so that the conservation degree can be interpreted as counting the configurations which satisfy (5).

Finally, the analysis in the elementary rule space of the frequency of occurrence of the equations in (14) whose logical value is true, shows that such a distribution is not constant, thus suggesting the idea of probing the effect of adding weights to each equation. However, a formulation of γ as such leads to different values among rules of the same equivalence class, a situation that is not appropriate.

3 Empirical conservation degree for one-dimensional rules

An empirical measure for the conservation degree must satisfy the facts that, according to (13), number-conserving rules should remain number-conserving in respect to the measure, and rules in the same class of dynamical equivalence should have the same empirical conservation degree. Accordingly, let $S \in \mathcal{Q}^L$, $L \in \mathbb{Z}^+$ be a configuration. If f is a number-conserving CA rule, then:

$$\sum_{i=1}^{L} s(i,t) = \sum_{i=1}^{L} s(i,t+1) \forall t \in \mathbb{Z}^+. \tag{15}$$

From (15), for each configuration we can define

$$\Theta_S = \{\sigma_t : \sigma_t = \sum_{i=1}^{L} s(i,t), t \in \mathbb{Z}^+\}, \tag{16}$$

from which, f is number-conserving if, and only if, $|\Theta| = 1$, $\forall S \in \mathcal{Q}^L$, $L \in \mathbb{N}$ and $t \in \mathbb{Z}^+$. Hence, a candidate for the empirical conservation degree can be defined as

$$\gamma_e = \frac{1}{L} \sum_{i=1}^{L} \frac{|\{S \in Q^i : |\Theta_S| = 1\}|}{q^i} \tag{17}$$

which represents the normalised sum of the proportions of number-conserving configurations for each length from 1 to L and fixed $t = T$ (i.e., the configurations that satisfy (15) up to a certain $t = T$). The values of γ_e for rules of the elementary space are depicted in Fig. 4.

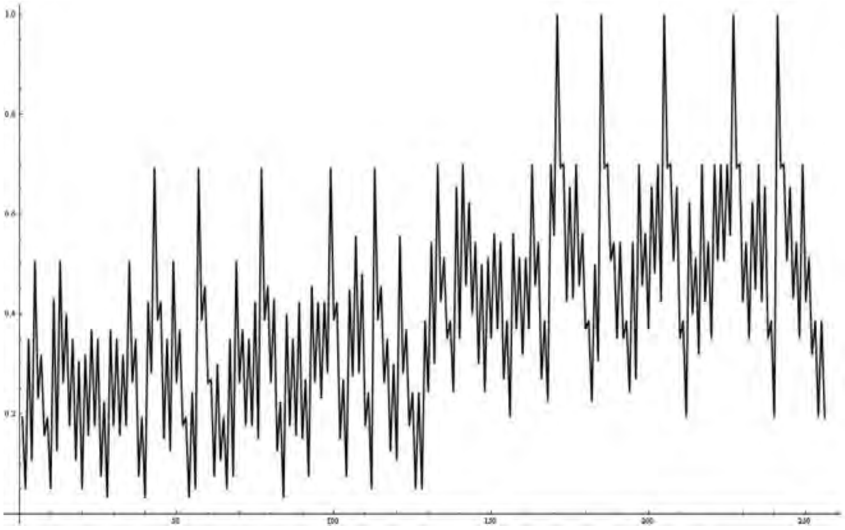


Fig. 4. Empirical conservation degree γe for L=5 and T=3 for all elementary rules, in order of their number.

Notice in Fig. 4 that there are 5 rules with $\gamma = 1.0$, corresponding to the elementary number-conserving rules 170, 184, 226 and 240, and 4 rules (27, 39, 53 and 83, all of them, by the way, belonging to the same dynamical equivalence class) with $\gamma = 0.03125$, the minimal value of conservation degree in the elementary space, since the normalisation of the sum of the proportions of number-conserving configurations from length 1 to L and fixed $t = T$ for these rules are also minimal.

Considering again $L = 5$ and $T = 3$ and then ploting γ and γ_e in the same coordinate system, Fig. 5 is obtained. Notice that, unless otherwise stated, in the following figures only the representative rule of each equivalence class of the elementary space [2] is considered, where the representative rule is the one possessing the smallest rule number; also, the points in the x-axis correspond to the ordinal position (in ascending order) of the representative rule, and not the rule number itself. As such, rule number 30 for instance, corresponds to the 27th position along the x-axis.

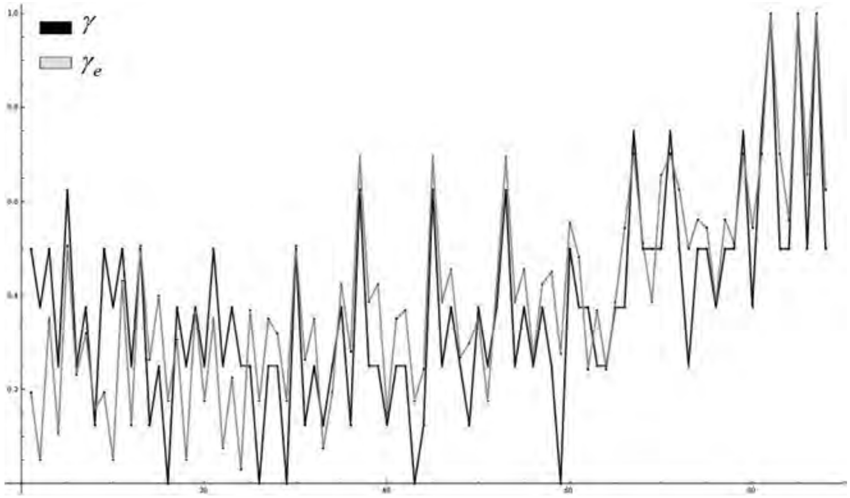


Fig. 5. Theoretical conservation degree γ and empirical conservation degree γ_e ($L = 5$ and $T = 3$), for all elementary classes of dynamical equivalence.

A possible explanation for the difference in the curves plotted in Fig. 5 may be the fact that, since γ depends on the conservation of the configuration of the form $(0, 0, x_1, x_2, x_3)$ and γ_e depends on the conservation of all configurations with size from length 1 up to 5, their values can be distinct.

Numerical experiments suggest that γ_e converges quickly as a function of t (on average for $t = 3$). However, for $L \in \{1, 2, \ldots, 30\} \cup \{40, 50, \ldots, 150\}$ (an approximation to $L \to \infty$) and $T = 10$, the situation displayed in Fig. 6 is typically observed, meaning that:

$$\lim_{L\to\infty} \gamma_e = \begin{cases} 1, \text{if} f \text{is number-conserving} \\ 0, \text{otherwise} \end{cases} \tag{18}$$

In fact, if f is number-conserving, then:

$$|\{S \in Q^i : \Theta_S = 1\}| = q^i \Rightarrow \lim_{L\to\infty} \gamma_e = 1 \tag{19}$$

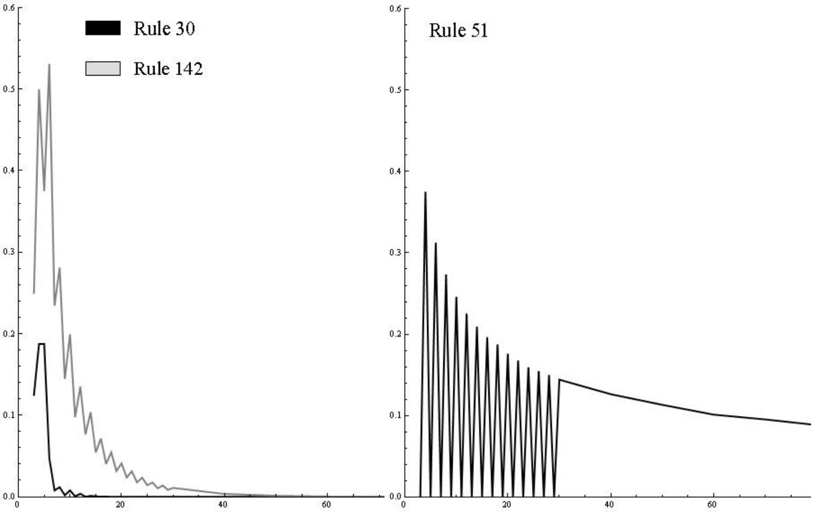


Fig. 6. Average amount of number-conserving configurations for elementary rules 30, 51 and 142 for every configuration length. All configurations were considered for $t = 10$ and $L = 16$, and random samples of 100,000 distinct configurations for $L > 16$.

As Fig. 6 makes it evident, on average the amount of number-conserving configurations decreases as the configuration lengths increase. Similar limit behaviour was observed for all 88 representative rules of the elementary space equivalence classes, even though three basic patterns of transients could be distinguished: non-monotonic convergence (as depicted in Fig. 6, for rules 30 and 142); monotonic convergence (as happened for rules 0, 2 and 132); and slow convergence with subsequent jumps from 0 to the calculated value of γ_e, for even and odd values of L, respectively (as in the case of rules 15, 23 and 51, the latter displayed in Fig. 6).

Although (6) provides necessary and sufficient conditions for deciding on the number-conservation property for a one-dimensional CA rule, an analogous condition has not yet been established with respect to the conservation degree of one-dimensional CA rules. However, considering that the conservation degree of a one-dimensional CA is defined as the normalisation of the sum of the number-conserving configurations from lengths 1 to L, the quantity becomes dependent on the length of the configuration at issue, so that, for large L, it tends to zero. Obviously, the computational cost for calculating the empirical conservation degree may become relatively high due to the exponential growth of configurations that should be considered.

3.1 Empirical conservation degree of one-dimensional Rules: An alternative approach

Since the previous definition of empirical conservation degree depends on the length of the configurations and converges to zero for all elementary rules, it cannot be an useful measure. In order to amend that, a new definition is proposed below.

Recalling the conditions that have to be met by an empirical measure of conservation degree, as expressed at the onset of this section, let $\gamma_{L,T}$ be defined as the arithmetic mean of the measure $|s(i,0) - s(i,t)|$ for all $i \in \{1,\ldots,L\}$ and $t \in \{1,\ldots,T\}$, as follows:

$$\gamma_{L,T} = \frac{1}{LT}\sum_{t=1}^{T} |\sum_{i=1}^{L}(s(i,0) - s(i,t))| \tag{20}$$

Now, based on the latter, another form for the empirical conservation degree (γ'_e) can be defined, as expressed by the difference between 1 and the mean of $\gamma_{L,T}$ over all configuration lengths $L \in \mathbb{N}$, in $T \in \mathbb{Z}^+$ iterations, for fixed L and T. Mathematically, the new quantity then becomes

$$\gamma'_e = 1 - \frac{1}{q^L}\sum_{S \in Q^L} \gamma_{L,T} \tag{21}$$

from which, if f is number-conserving, then

$$\sum_{i=1}^{L}(s(i,0) - s(i,t)) = 0 \Rightarrow \gamma'_e = 1 \tag{22}$$

Numerical experiments suggest that γ'_e converges quickly as a function of T (on average for $T = 15$), but much slower as a function of L. However, as verified for $L \in \{1,2,\ldots,30\} \cup \{40,50,\ldots,150\}$, all 88 representative rules for the equivalence classes in the elementary space converge for sufficiently large L. For instance, on Fig. 7 the asymptotic behaviour of γ'_e for elementary rules 0, 146 and 154 can be realised. These rules present, respectively, the minimal value ($\gamma'_e = 0.5$) of the space, the maximum possible value ($\gamma'_e = 0.976$) for a non-conservative rule, and an intermediate value ($\gamma'_e = 0.772$), that is close to the arithmetic mean of the latter two.

Spatio-temporal analyses of the elementary rules suggest that γ'_e is related indeed to a certain capacity of the rule to maintain density constant in relation to the initial configuration, which, as expected, can be interpreted as a measure of conservation degree. As illustration, the values of Θ_S for rules 0, 146 and 154 are presented in Fig. 8, for each configuration of length $L = 4$ and $T = 3$. Notice for rule 0, that the difference between the first term of each quadruplet and the last one is maximum, meaning that the rule capacity to maintain constant its density in relation to the initial configuration is minimal. Such behaviour is expected, since rule 0 maps any configuration to the one composed of all 0s. It also explains

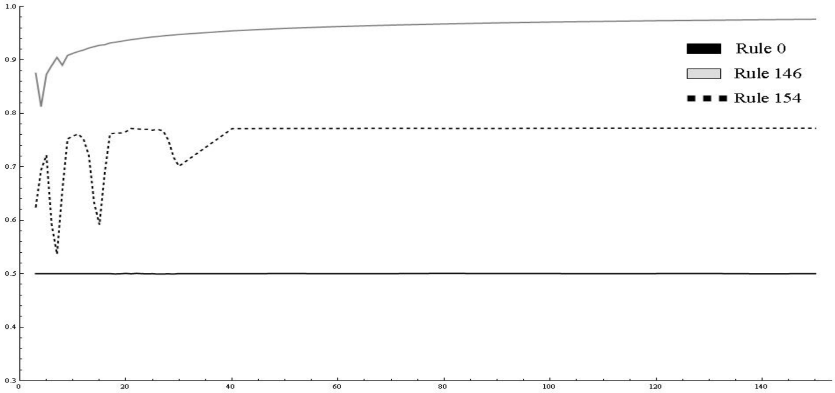
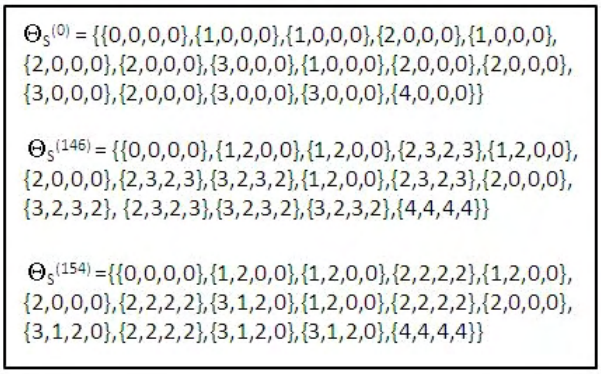


Fig. 7. Empirical conservation degree γ'_e of elementary rules 0, 146 and 154, for various configuration lengths. The graph refers to all configurations for $T = 10$ and $L = 16$, and random samples of 50,000 different configurations for $L > 16$.



$\Theta_S{}^{(0)}$ = {{0,0,0,0},{1,0,0,0},{1,0,0,0},{2,0,0,0},{1,0,0,0},
{2,0,0,0},{2,0,0,0},{3,0,0,0},{1,0,0,0},{2,0,0,0},{2,0,0,0},
{3,0,0,0},{2,0,0,0},{3,0,0,0},{3,0,0,0},{4,0,0,0}}

$\Theta_S{}^{(146)}$ = {{0,0,0,0},{1,2,0,0},{1,2,0,0},{2,3,2,3},{1,2,0,0},
{2,0,0,0},{2,3,2,3},{3,2,3,2},{1,2,0,0},{2,3,2,3},{2,0,0,0},
{3,2,3,2}, {2,3,2,3},{3,2,3,2},{3,2,3,2},{4,4,4,4}}

$\Theta_S{}^{(154)}$ ={{0,0,0,0},{1,2,0,0},{1,2,0,0},{2,2,2,2},{1,2,0,0},
{2,0,0,0},{2,2,2,2},{3,1,2,0},{1,2,0,0},{2,2,2,2},{2,0,0,0},
{3,1,2,0},{2,2,2,2},{3,1,2,0},{3,1,2,0},{4,4,4,4}}

Fig. 8. Θ_S for elementary rules 0, 146 and 154. Each quadruplet represents the values of Θ_S for each configuration of length $L = 4$ and $T = 3$.

the quick convergence observed for rule 0, shown in Fig. 7. Conversely, for rule 146, the difference between the first term of each quadruplet and the last one is close to 0 for almost all quadruplets, indicating the capacity of the rule to maintain constant its density in relation to the initial configuration tends to be maximum. Finally, both previous patterns can be observed for rule 154.

Figure 9 shows the convergence of $\gamma'_e$for all representative rules of the dynamical equivalence classes in the elementary space.

According to [2], rules can be categorised into four classes of dynamical behaviour: Class-I (initial configurations converge to a homogeneous configuration), Class-II (configurations converge to a fixed-point or a cycle), Class-III (a chaotic or pseudo-random behaviour is typically observed), and Class-IV (configurations

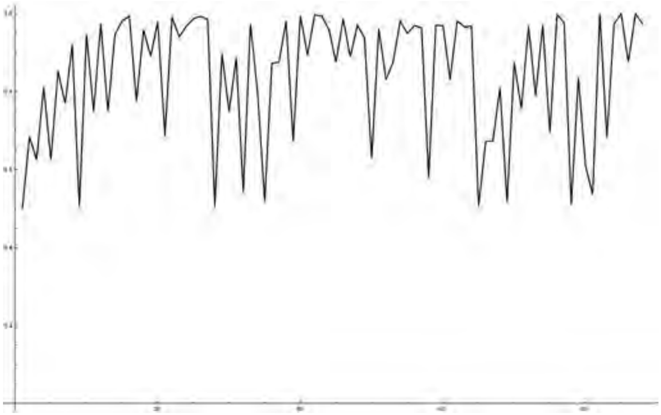


Fig. 9. Empirical conservation degree γ'_e for each representative of the dynamical equivalence classes in the elementary space. The graph refers to all configurations for $T = 10$ and $L = 16$, and random samples of 50,000 different configurations for $L > 16$.

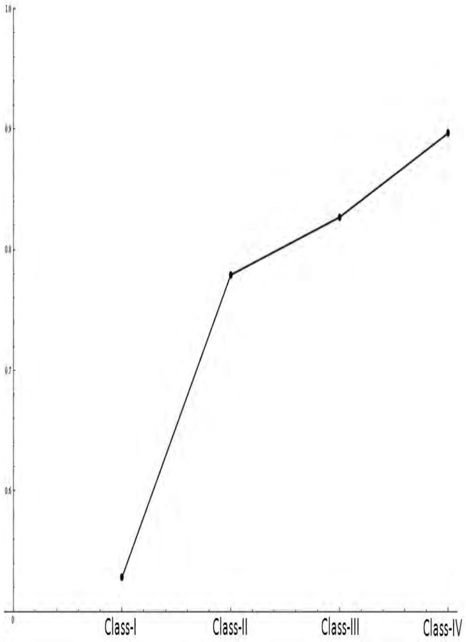


Fig. 10. Arithmetic mean of γ'_e for rules in each equivalence class of dynamical behaviour.

often display complex patterns of localised structures). So, considering this classification, the arithmetic mean of γ_e' for rules belonging to each of these classes can be noticed in Fig. 10.

According to Wolfram's classification scheme, lower values of γ_e' for Class-I rules are expected, since almost all configurations converge to a homogeneous configuration; hence, the difference between the first term and the last one of their Θ_S becomes maximum. For Class-II rules, higher values of γ_e' are expected than those for Class-I, since some configurations may converge to homogeneous configurations or cycles of number-conserving configurations. For Class-III and Class-IV rules, Θ_S analyses are not so evident. In fact, while all elementary number-conserving rules belong to Class-II, the arithmetic mean of γ_e' for Class-IV has the greatest value. So, for the elementary space, complex rules are, in average, the most conservative ones.

3.2 Further analyses of γ_e'

The possibility of establishing a relationship between γ and γ_e' is appealing in that it might provide an indirect analytical method for the calculation of γ_e', which is independent on the configuration lengths. Comparing such parameters it happens that rules with the same value of γ may have different values of γ_e' and vice versa. Indeed, the granularity presented by γ'e is bigger than that presented by γ; for example, $n = 3$ and $q = 2$ entails $|\Phi \cup \Phi_C| = 8$, implying that γ can take on no more than 9 different values (in fact, there are exactly 8 values), while γ_e' takes on 86 different values.

In order to overcome the distinct granularities of the two measures and attempt a compatible match between them, let $\delta_{\gamma_e} : \Gamma' \rightarrow \Gamma$ be a function that maps the values of γ_e' onto the smaller set defined by the values of γ, where Γ' and Γ are the image sets of γ_e' and γ, respectively, such that:

$$\delta_{\gamma_e}(x) = \begin{cases} 1, \text{if } x = 1 \\ \gamma_i, \text{if } \gamma_i \leq x < \gamma_{i+1} \end{cases} \tag{23}$$

In Fig. 11 the overall relationship between δ_{γ_e} and γ can be noticed. From the graph it can be drawn that $\delta_{\gamma_e} = 0.75$ for both rules 76 and 105, suggesting that these rules should display similar behaviours from the perspective of that quantity. However, analyses of Θ_S for these rules showed this does not hold true. Fig. 12 presents Θ_S values for rules 76 and 105, respectively, for each configuration of length $L = 4$ and $T = 3$. Such behaviour is expected, since $\gamma_e'^{(76)} = 0.875 < 0.970 = \gamma_e'^{(105)}$, indicating a higher average size of $\gamma_{L,T}$ for rule 76 and, therefore, a lower empirical degree of conservation γ_e'. From a different pespective, Fig. 12 clearly shows that there are more number-conserving configurations associated to rule 105 than to rule 76, once again pointing to the fact that $\gamma_e'^{(105)} > \gamma_e'^{(76)}$. In conclusion, the idea of trying to match δ_{γ_e} and γ does not help as a conceptual tool for linking the analytical conservation degree measure γ and the empirical counterpart γ_e'.

Finally, a relationship between γ_e' and Langton's λ activity parameter [6] can be noticed, as illustrated in Fig. 13. This relationship is due to the fact that

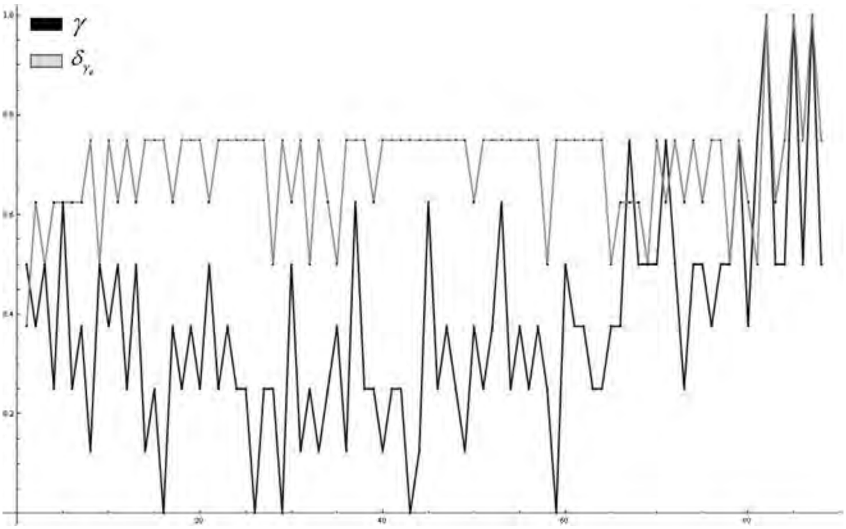
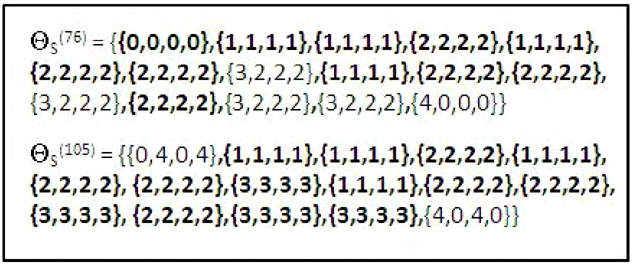


Fig. 11. Empirical conservation degrees γ and δ_{γ_e} (the mapped version of γ'_e onto γ), for the elementary space dynamical equivalence classes (under the same conditions of Fig. 5).

extreme values of λ, either high or low, can bring in greater variation of (20) and, consequently, also of γ'_e.



$\Theta_S^{(76)}$ = {**{0,0,0,0}**,**{1,1,1,1}**,**{1,1,1,1}**,**{2,2,2,2}**,**{1,1,1,1}**,
{2,2,2,2},**{2,2,2,2}**,{3,2,2,2},**{1,1,1,1}**,**{2,2,2,2}**,**{2,2,2,2}**,
{3,2,2,2},**{2,2,2,2}**,{3,2,2,2},{3,2,2,2},{4,0,0,0}}

$\Theta_S^{(105)}$ = {{0,4,0,4},**{1,1,1,1}**,**{1,1,1,1}**,**{2,2,2,2}**,**{1,1,1,1}**,
{2,2,2,2}, **{2,2,2,2}**,**{3,3,3,3}**,**{1,1,1,1}**,**{2,2,2,2}**,**{2,2,2,2}**,
{3,3,3,3}, **{2,2,2,2}**,**{3,3,3,3}**,**{3,3,3,3}**,{4,0,4,0}}

Fig. 12. Θ_S for elementary rules 76 and 105. The quadruplets in bold face represent the values of Θ_S for number-conserving configurations of length $L = 4$ and $T = 3$.

4 Conclusion

A definition was proposed here for the conservation degree of one-dimensional CA rules, based on Boccara-Fukś conditions. A theoretical measure of conserva-

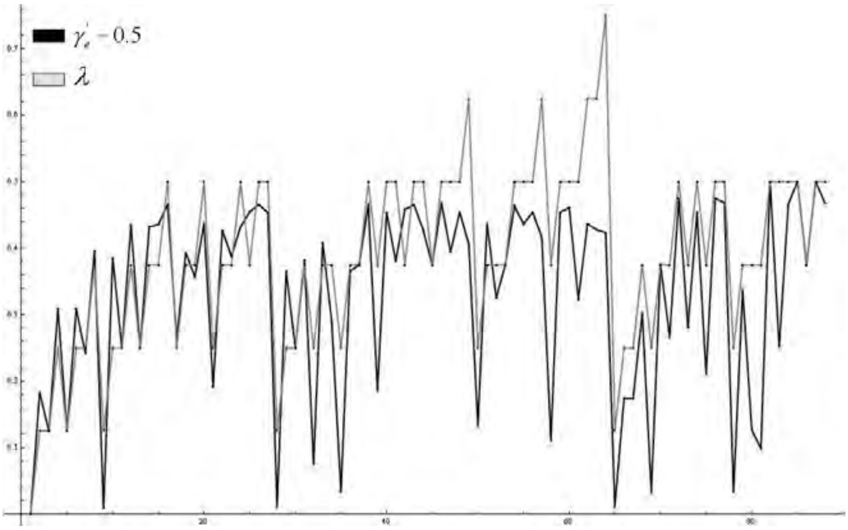


Fig. 13. The value γ_e'–0.5 and Langton's λ parameter, for all elementary classes of dynamical equivalence (under the same conditions of Fig. 5).

tion degree was defined, which is consistent among the rules of the same class of dynamical equivalence, and two empirical conservation degree measures have also been proposed.

The first empirical measure showed that the mean of the amount of number-conserving configurations tends to decrease as the configuration lengths increase; consequently, assuming that conservation involves the amount of number-conserving configurations, such a measure would not depend only on the CA rule, but also on the configuration lengths considered. Numerical experiments showed that for all rules of the elementary space, their conservation degrees tend to zero as the configuration lengths increase.

The second empirical measure defined is based on the capacity that the rule has to maintain the quantities in relation to the initial configuration. Numerical experiments showed that for all rules in the elementary space, their conservation degrees converge for sufficiently large configuration lengths, suggesting the measure is related to the rule global dynamics. In fact, some clear relations could be established between Wolfram's classes of dynamical behaviour and the mean value of the empirical conservation degree for each class.

In the absence of a definition for the theoretical conservation degree some attempts were made in order to relate the theoretical and empirical results obtained; however, no relations were found. Numerical experiments indicate that both empirical conservation degrees preserve their qualitative behaviours when other rule spaces are considered; however, more research is needed not only in this direction, but also in the theoretical formulation of conservation degree equivalent to the last one defined.

Acknowledgements

This work has been possible due to research grants provided to PPBO by Wolfram Research (Mathematica Academic Grant No 1149), FAPESP — Fundação de Amparo à Pesquisa do Estado de São Paulo (Proc. 2005/04696-3), and MackPesquisa — Fundação Mackenzie de Pesquisa (Edital 2007). A.Schranko also thanks financial support from CAPES — Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, and MackPesquisa — Fundação Mackenzie de Pesquisa.

References

[1] Wuensche, A., Lesser M. J.: "The Global Dynamics of Cellular Automata". Santa Fe Institute Studies in the Sciences of Complexity, Addison -Wesley (1992)

[2] Wolfram, S.: *Cellular Automata and Complexity.* Addison-Wesley. (1994)

[3] Nagel, K., Schreckenberg, M.: "A Cellular Automaton Model for Freeway Traffic". *Journal de Physique* I 2, pp. 2221-2229 (1992)

[4] Hattori, T., Takesue, S.: "Additive conserved quantities in discrete-time lattice dynamical systems". *Physica D 49*, pp. 295-322 (1991)

[5] Boccara, N., Fukś, H.: "Number-Conserving Cellular Automaton Rules". *Fundamenta Informaticae* 52, pp. 1-13 (2002)

[6] Langton, C. "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation". *Physica D 42*, pp. 12-37 (1990)

A family of smallest symmetrical four-state firing squad synchronization protocols for one-dimensional ring cellular automata

Hiroshi Umeo[1], Naoki Kamikawa[1], and Jean-Baptiste Yunès[2]

[1] Univ. of Osaka Electro-Communication,
Neyagawa-shi, Hastu-cho, 18-8, Osaka, 572-8530, Japan
umeo@cyt.osakac.ac.jp
[2] LIAFA – Universite Paris 7 Denis Diderot.
175, rue du chevaleret. 75013 Paris - France
Jean-Baptiste.Yunes@liafa.jussieu.fr

Abstract. An existence or non-existence of five-state firing squad synchronization protocol has been a longstanding and famous open problem for a long time. In this paper, we answer partially to this problem by proposing a family of smallest four-state firing squad synchronization protocols that can synchronize any one-dimensional ring cellular array of length $n = 2^k$ for any positive integer k. The number $four$ is the smallest one in the class of synchronization protocols proposed so far.

1 Introduction

We study a synchronization problem that gives a finite-state protocol for synchronizing a large scale of cellular automata. The synchronization in cellular automata has been known as a firing squad synchronization problem since its development, in which it was originally proposed by J. Myhill in Moore [1964] to synchronize all parts of self-reproducing cellular automata. The firing squad synchronization problem has been studied extensively for more than 40 years [1-12]. The optimum-time (i.e., $(2n-2)$-step) synchronization algorithm was devised first by Goto [1962] for one-dimensional array of length n. The algorithm needed many thousands of internal states for its realization. Afterwards, Waksman [1966], Balzer [1967], Gerken [1987] and Mazoyer [1987] also developed an optimum-time algorithm and reduced the number of states realizing the algorithm, each with 16, 8, 7 and 6 states. On the other hand, Balzer [1967], Sanders [1994] and Berthiaume et al. [2004] have shown that there exists no four-state synchronization algorithm. Thus, an existence or non-existence of five-state firing squad synchronization protocol has been a longstanding and famous open problem for a long time. Umeo and Yanagihara [2007] initiated an investigation on the FSSP solutions that can synchronize some infinite set of arrays, but not all, and presented a five-state $3n + O(1)$ step algorithm that can synchronize any one-dimensional cellular array of length $n = 2^k$ for any positive integer k

in $3n - 3$ steps. Recently, Yunès [2008] and Umeo, Yunès, and Kamikawa [2008] developed 4-state protocols based on Wolfram's rule 60 and 150.

In this paper, we answer partially to the problem by proposing a family of smallest four-state firing squad synchronization protocols that can synchronize any one-dimensional ring cellular array of length $n = 2^k$ for any positive integer k. The number $four$ is the smallest one in the class of synchronization protocols proposed so far. Due to the space availability we only give informal descriptions of the family of the four-state solutions.

2 Firing squad synchronization problem

2.1 Definition of the firing squad synchronization problem

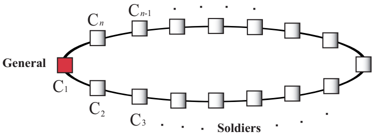


Fig. 1. One-dimensional ring cellular automaton.

The firing squad synchronization problem is formalized in terms of the model of cellular automata. Figure 1 shows a finite one-dimensional ring cellular array consisting of n cells, denoted by C_i, where $1 \leq i \leq n$. All cells are identical finite state automata. The array operates in lock-step mode such that the next state of each cell is determined by both its own present state and the present states of its right and left neighbors. All cells (*soldiers*), except one cell, are initially in the *quiescent* state at time $t = 0$ and have the property whereby the next state of a quiescent cell having quiescent neighbors is the quiescent state. At time $t = 0$ the cell C_1 (*general*) is in the *fire-when-ready* state, which is an initiation signal to the array.

The firing squad synchronization problem is stated as follows: Given an array of n identical cellular automata, including a *general* cell which is activated at time $t = 0$, we want to give the description (state set and next-state transition function) of the automata so that, *at some future time*, all of the cells will *simultaneously* and, *for the first time*, enter a special *firing* state. The set of states and the next-state transition function must be independent of n. Without loss of generality, we assume $n \geq 2$. The tricky part of the problem is that the same kind of soldier having a fixed number of states must be synchronized, regardless of the length n of the array. One has to note that any solution in the original problem is to synchronize any array of length greater than two. We call it **full** solution. On the other hand, Umeo and Yanagihara [2007] initiated an investigation on the FSSP solutions that can synchronize some infinite set of arrays, but not all, and presented a five-state $3n + O(1)$ step algorithm that can synchronize any one-dimensional cellular array of length $n = 2^k$ for any positive integer k in $3n - 3$ steps. Recently, Yunès [2008] and Umeo, Yunès, and Kamikawa [2008] developed 4-state protocols based on Wolfram's rule 150 and 60. We call such protocol **partial** solution.

We summarize recent results on state lower bounds in FSSP. In the statements in the theorems below the term *open-ring* means a conventional one-dimensional array with a general at one end.

[Theorem 1]$^{\text{Berthiaume, Bittner, Perkovic, Settle, and Simon [2004]}}$ (State Lower Bound) There is no 3-state *full* solution to the firing squad synchronization problem for the ring.

[Theorem 2]$^{\text{Berthiaume, Bittner, Perkovic, Settle, and Simon [2004]}}$ (State Lower Bound) There is no 4-state, symmetric, minimal-time *full* solution to the firing squad synchronization problem for the ring.

[Theorem 3]$^{\text{Balzer [1967], Sanders [1994]}}$ (State Lower Bound) There is no 4-state *full* solution to the firing squad synchronization problem for the open-ring.

[Theorem 4]$^{\text{Umeo, Yunes, and Kamikawa [2008], Yunes [2008]}}$ There exist 4-state *partial* solutions to the firing squad synchronization problem for the open-ring.

[Theorem 5]$^{\text{Yunes [2008]}}$ There exist no 3-state *partial* solutions to the firing squad synchronization problem for the open-ring.

In this paper we will establish the following theorems with a help of computer investigation.

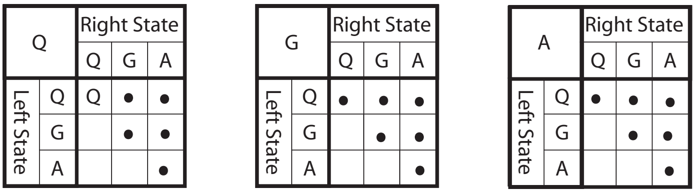
[Theorem 6] There is no 3-state *partial* solution to the firing squad synchronization problem for the ring.

[Theorem 7] There exist seventeen symmetrical 4-state partial solutions to the firing squad synchronization problem for the ring of length $n = 2^k$ for any positive integer k.

3 A quest for four-state symmetrical partial solutions for rings

3.1 Four-state symmetrical ring cellular automata

Let $\mathcal{M}$ be a four-state ring cellular automaton $\mathcal{M} = \{\mathcal{Q}, \delta\}$, where $\mathcal{Q}$ is an internal state set and δ is a transition function such that $\mathcal{Q} = \{A, F, G, Q\}$, $\delta : \mathcal{Q}^3 \rightarrow \mathcal{Q}$. Without loss of generality, we assume that Q is a quiescent state with a property $\delta(Q, Q, Q) = Q$, A and G are auxiliary states and F is the *firing* state, respectively. One of the auxiliary states is assumed to be the general state. The initial configuration is either $G\overbrace{QQ, ..., Q}^{n-1}$ or $A\overbrace{QQ, ..., Q}^{n-1}$ for $n \geq 2$, depending on the general state G or A, respectively.



Q		Right State		
		Q	G	A
Left State	Q	Q	●	●
	G		●	●
	A			●

G		Right State		
		Q	G	A
Left State	Q	●	●	●
	G		●	●
	A			●

A		Right State		
		Q	G	A
Left State	Q	●	●	●
	G		●	●
	A			●

Fig. 2. Symmetrical four-state transition table.

3.2 A computer investigation into four-state symmetrical FSSP solutions for rings

We consider only symmetrical four-state ring cellular automata. Figure 2 is a symmetrical four-state transition table, where a symbol ● shows a possible state in $\mathcal{Q} = \{A, F, G, Q\}$. A blank symbol in the table is uniquely determined by the symmetrical property such that $\delta(x, y, z) = \delta(z, y, x)$, for any state x, y, z in $\mathcal{Q}$. To define a symmetrical transition table, it is sufficient to specify the entries shown by ● in Fig. 2. Thus, we have totally 4^{17} possible transition rules. This is no problem on today's computer. We make a computer investigation into the transition rule set that might yield an FSSP solution. Our searching strategy is a very simple *generate-and-test* method described below:

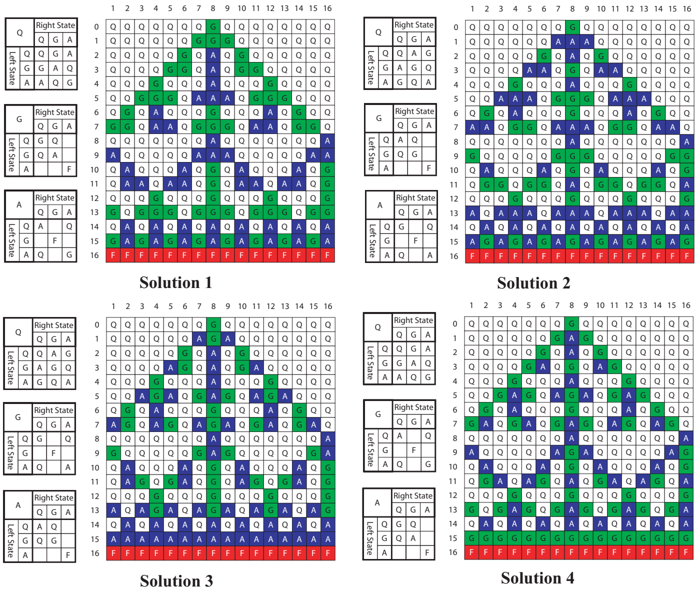
1. **Generate** a symmetrical 4-state transition table.
2. **Compute** the configuration of each transition table for a ring of length n, starting from an initial configuration: $G\overbrace{Q, ..., Q}^{n-1}$, such that $2 \leq n \leq 32$ and **check** whether each rule yields a synchronized configuration: $\overbrace{FF, ..., F}^{n}$ during the time t such that $n \leq t \leq 4n$ and the state F never appears in the configurations before.

   Observation 1: We have found that there exist many thousands of possible partial solutions satisfying the conditions above. Exactly, we have got 6412 transition rule sets that yield successful synchronized configurations. One of the biggest reasons for the large number of possible solutions is that most of them include some redundant entries that give no influence to the computation.

3. **Remove** all redundant entries in each solution.
4. **Compare and classify** those valid solutions into small groups.

   Observation 2: Eventually we have got seventeen solutions that consist of four optimum-time solutions, four nearly-optimum-time solutions, and nine non-optimum-time solutions. Surprisingly, all of the solutions obtained can synchronize rings of length $n = 2^k$ for any positive integer k.

The outline of those solutions will be described in the next section.



Solution 1

Solution 2

Solution 3

Solution 4

Fig. 3. Transition tables and snapshots on 16 cells for the Solutions 1, 2, 3, and 4.

4 Four-state solutions

4.1 Optimum-time solutions

It has been shown by Berthiaume et al. [2004] that there exists no algorithm that can synchronize any ring of length n in less than n steps.

[Theorem 8]Berthiaume, Bittner, Perkovic, Settle, and Simon [2004] The minimum time in which the firing squad synchronization could occur is no earlier than n steps for any ring of length n.

We have got four optimum-time *partial* solutions operating in exactly n steps. Figure 3 shows the transition rules and snapshots on 16 cells for Solutions 1, 2, 3 and 4. It is noted that both of the states G and A can be an initial general state in each solution without introducing any additional transition rules. Let $T_{i,G}(n)$, $T_{i,A}(n)$ be time complexity of Solution i for synchronizing a ring CA of length n with an initial general in state G, A, respectively. We get the following observation.

Observation 3: For any i such that $1 \leq i \leq 4$, $T_{i,G}(n) = T_{i,A}(n) = n$.

In Table 4.1 we give the time complexity and number of transition rules for each solutions.

Table 1. Time complexity and number of transition rules for Solutions 1, 2, 3 and 4.

Solutions	Time complexity	# of transition rules	Notes
Solution 1	$T_{1,G}(n) = T_{1,A}(n) = n$	19	
Solution 2	$T_{2,G}(n) = T_{2,A}(n) = n$	19	
Solution 3	$T_{3,G}(n) = T_{3,A}(n) = n$	19	
Solution 4	$T_{4,G}(n) = T_{4,A}(n) = n$	19	

4.2 Nearly optimum-time solutions

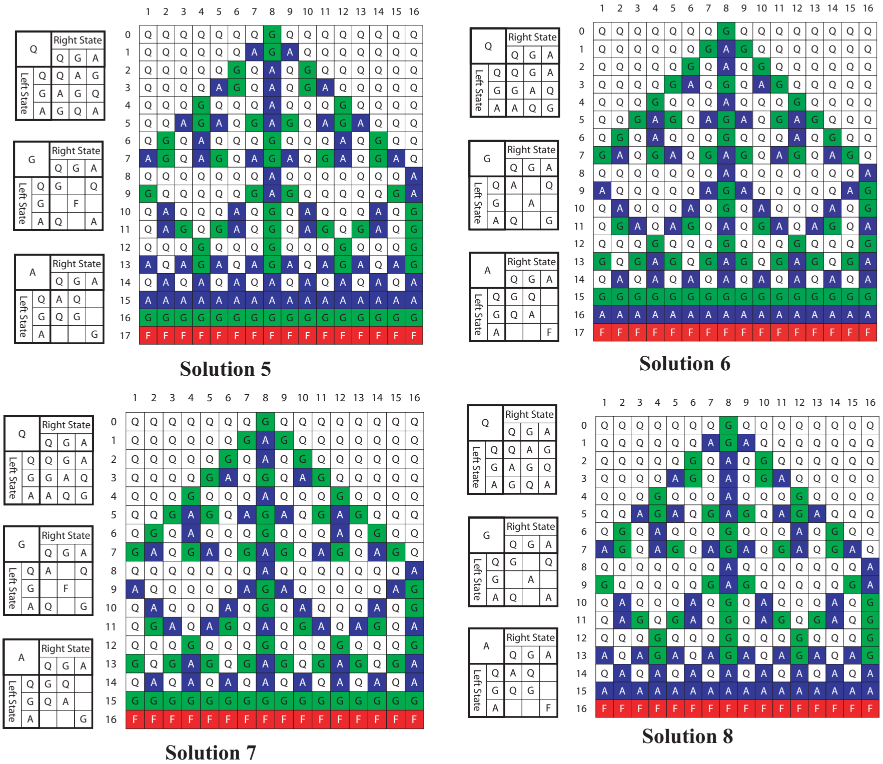
We have got four nearly optimum-time Solutions 5, 6, 7, and 8, each can synchronize rings of length $n = 2^k$ in n or $n+1$ steps. Figure 4 shows the transition rules and snapshots on 16 cells for Solutions 5, 6, 7 and 8. It is noted that both of the states G and A can be an initial general state in each solution without introducing any additional transition rules. Those solutions have an interesting property on synchronization time such that:

Observation 4: $T_{5,G}(n) = \begin{cases} n & k : odd \\ n+1 & k : even \end{cases}$ $T_{5,A}(n) = \begin{cases} n+1 & k : odd \\ n & k : even. \end{cases}$

Similar observations can be made for the Solutions 6, 7 and 8. In Table 2 we give the time complexity and number of transition rules for each solution.

4.3 Non-optimum-time solutions

Here we give nine non-optimum-time Solutions 9, 10, 11, 12, 13, 14, 15, 16 and 17, each can synchronize rings of length $n = 2^k$ in $3n/2 - O(1)$ or $2n - O(1)$ steps. Figure 5 shows the transition rules and snapshots on 16 cells for Solutions 9, 10, 11 and 12 and Figure 6 shows the transition rules and snapshots on 16 cells for Solutions 13, 13, 15, 16 and 17. In Table 3 we give the time complexity and number of transition rules for each solution. In those solutions, except the Solution 11, It is observed that both of the states G and A can be an initial general state in each solution without introducing any additional transition rules. The Solution 11 with an initial general in state A cannot be a partial solution as it is.



Solution 5

Solution 6

Solution 7

Solution 8

Fig. 4. Transition tables and snapshots on 16 cells for the Solutions 5, 6, 7, and 8.

Table 2. Time complexity and number of transition rules for Solutions 5, 6, 7 and 8.

Solutions	Time complexity	# of transition rules	Notes
Solution 5	$T_{5,G}(n) = \begin{cases} n & k : odd \\ n+1 & k : even \end{cases}$	19	$n = 2^k$
Solution 6	$T_{6,G}(n) = \begin{cases} n & k : odd \\ n+1 & k : even \end{cases}$	19	$n = 2^k$
Solution 7	$T_{7,G}(n) = \begin{cases} n+1 & k : odd \\ n & k : even \end{cases}$	19	$n = 2^k$
Solution 8	$T_{8,G}(n) = \begin{cases} n+1 & k : odd \\ n & k : even \end{cases}$	19	$n = 2^k$

Q		Right State		
		Q	G	A
Left State	Q	Q	G	A
	G	G	A	Q
	A	A	Q	Q

G		Right State		
		Q	G	A
Left State	Q	A		Q
	G		F	
	A	Q		G

A		Right State		
		Q	G	A
Left State	Q	G	Q	
	G	Q	A	
	A			G

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	Q
1	Q	Q	Q	Q	Q	Q	G	A	G	Q	Q	Q	Q	Q	Q	Q
2	Q	Q	Q	Q	Q	G	Q	A	Q	G	Q	Q	Q	Q	Q	Q
3	Q	Q	Q	Q	G	A	Q	G	Q	A	G	Q	Q	Q	Q	Q
4	Q	Q	Q	G	Q	Q	Q	A	Q	Q	Q	G	Q	Q	Q	Q
5	Q	Q	G	A	G	Q	A	G	A	Q	G	A	G	Q	Q	Q
6	Q	G	Q	A	Q	Q	Q	G	Q	Q	Q	A	Q	G	Q	Q
7	G	A	Q	G	A	Q	G	A	G	Q	A	G	Q	A	G	Q
8	Q	Q	Q	Q	Q	Q	Q	A	Q	Q	Q	Q	Q	Q	Q	A
9	A	Q	Q	Q	Q	Q	A	G	A	Q	Q	Q	Q	Q	A	G
10	Q	A	Q	Q	Q	A	Q	G	Q	A	Q	Q	Q	A	Q	G
11	Q	G	A	Q	A	G	Q	A	Q	G	A	Q	A	G	Q	A
12	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	G
13	G	Q	Q	Q	Q	Q	G	A	G	Q	Q	Q	Q	Q	G	A
14	Q	G	Q	Q	Q	G	Q	A	Q	G	Q	Q	Q	G	Q	A
15	Q	A	G	Q	G	A	Q	G	Q	A	G	Q	G	A	Q	G
16	Q	Q	Q	A	Q	Q	Q	A	Q	Q	Q	A	Q	Q	Q	A
17	A	Q	A	G	A	Q	A	G	A	Q	A	G	A	Q	A	G
18	Q	Q	Q	G	Q	Q	Q	G	Q	Q	Q	G	Q	Q	Q	G
19	G	Q	G	A	G	Q	G	A	G	Q	G	A	G	Q	G	A
20	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A
21	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G
22	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
23	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G
24	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Solution 9

Q		Right State		
		Q	G	A
Left State	Q	Q	A	G
	G	A	Q	Q
	A	G	Q	A

G		Right State		
		Q	G	A
Left State	Q	A	Q	
	G	Q	G	
	A			F

A		Right State		
		Q	G	A
Left State	Q	G		Q
	G		F	
	A	Q		A

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	Q
1	Q	Q	Q	Q	Q	Q	A	A	A	Q	Q	Q	Q	Q	Q	Q
2	Q	Q	Q	Q	Q	G	Q	A	Q	G	Q	Q	Q	Q	Q	Q
3	Q	Q	Q	Q	A	A	Q	G	Q	A	A	Q	Q	Q	Q	Q
4	Q	Q	Q	G	Q	Q	Q	A	Q	Q	Q	G	Q	Q	Q	Q
5	Q	Q	A	A	A	Q	G	G	G	Q	A	A	A	Q	Q	Q
6	Q	G	Q	A	Q	Q	Q	G	Q	Q	Q	A	Q	G	Q	Q
7	A	A	Q	G	G	Q	A	A	A	Q	G	G	Q	A	A	Q
8	Q	Q	Q	Q	Q	Q	Q	A	Q	Q	Q	Q	Q	Q	Q	A
9	G	Q	Q	Q	Q	Q	G	G	G	Q	Q	Q	Q	Q	G	G
10	Q	A	Q	Q	Q	A	Q	G	Q	A	Q	Q	Q	A	Q	G
11	Q	G	G	Q	G	G	Q	A	Q	G	G	Q	G	G	Q	A
12	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	G
13	A	Q	Q	Q	Q	Q	A	A	A	Q	Q	Q	Q	Q	A	A
14	Q	G	Q	Q	Q	G	Q	A	Q	G	Q	Q	Q	G	Q	A
15	Q	A	A	Q	A	A	Q	G	Q	A	A	Q	A	A	Q	G
16	Q	Q	Q	A	Q	Q	Q	A	Q	Q	Q	A	Q	Q	Q	A
17	G	Q	G	G	G	Q	G	G	G	Q	G	G	G	Q	G	G
18	Q	Q	Q	G	Q	Q	Q	G	Q	Q	Q	G	Q	Q	Q	G
19	A	Q	A	A	A	Q	A	A	A	Q	A	A	A	Q	A	A
20	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A
21	A	G	A	G	A	G	A	G	A	G	A	G	A	G	A	G
22	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Solution 10

Q		Right State		
		Q	G	A
Left State	Q	Q	G	A
	G	G	Q	A
	A	A	A	G

G		Right State		
		Q	G	A
Left State	Q	A	Q	G
	G	Q	A	G
	A	G	G	Q

A		Right State		
		Q	G	A
Left State	Q	G	A	
	G	A	Q	Q
	A		Q	F

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	Q
1	Q	Q	Q	Q	Q	Q	G	A	G	Q	Q	Q	Q	Q	Q	Q
2	Q	Q	Q	Q	Q	G	G	Q	G	G	Q	Q	Q	Q	Q	Q
3	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q
4	Q	Q	Q	G	A	G	Q	Q	Q	G	A	G	Q	Q	Q	Q
5	Q	Q	G	G	Q	G	G	Q	G	G	Q	G	G	Q	Q	Q
6	Q	G	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	G	Q	Q
7	G	A	G	Q	Q	Q	Q	Q	Q	Q	Q	Q	G	A	G	Q
8	G	Q	G	G	Q	Q	Q	Q	Q	Q	Q	G	G	Q	G	Q
9	A	Q	Q	Q	G	Q	Q	Q	Q	Q	G	Q	Q	Q	A	Q
10	G	A	Q	G	A	G	Q	Q	Q	G	A	G	Q	A	G	G
11	G	A	A	G	Q	G	G	Q	G	G	Q	G	A	A	G	A
12	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q
13	Q	Q	G	A	G	Q	Q	Q	Q	Q	G	A	G	Q	Q	Q
14	Q	G	G	Q	G	G	Q	Q	Q	G	G	Q	G	G	Q	Q
15	G	Q	Q	Q	Q	Q	G	Q	G	Q	Q	Q	Q	Q	G	Q
16	A	G	Q	Q	Q	G	A	Q	A	G	Q	Q	Q	G	A	Q
17	A	G	G	Q	G	G	A	G	A	G	G	Q	G	G	A	G
18	Q	G	Q	Q	Q	G	Q	Q	Q	G	Q	Q	Q	G	Q	Q
19	G	A	G	Q	G	A	G	Q	G	A	G	Q	G	A	G	Q
20	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q
21	A	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A	Q
22	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	G
23	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
24	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Solution 11

Q		Right State		
		Q	G	A
Left State	Q	Q	G	A
	G	G	A	Q
	A	A	Q	Q

G		Right State		
		Q	G	A
Left State	Q	A		Q
	G			
	A	Q		G

A		Right State		
		Q	G	A
Left State	Q	G	Q	
	G	Q	A	
	A			F

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	Q
1	Q	Q	Q	Q	Q	Q	G	A	G	Q	Q	Q	Q	Q	Q	Q
2	Q	Q	Q	Q	Q	G	Q	A	Q	G	Q	Q	Q	Q	Q	Q
3	Q	Q	Q	Q	G	A	Q	G	Q	A	G	Q	Q	Q	Q	Q
4	Q	Q	Q	G	Q	Q	Q	A	Q	Q	Q	G	Q	Q	Q	Q
5	Q	Q	G	A	G	Q	A	G	A	Q	G	A	G	Q	Q	Q
6	Q	G	Q	A	Q	Q	Q	G	Q	Q	Q	A	Q	G	Q	Q
7	G	A	Q	G	A	Q	G	A	G	Q	A	G	Q	A	G	Q
8	Q	Q	Q	Q	Q	Q	Q	A	Q	Q	Q	Q	Q	Q	Q	A
9	A	Q	Q	Q	Q	Q	A	G	A	Q	Q	Q	Q	Q	A	G
10	Q	A	Q	Q	Q	A	Q	G	Q	A	Q	Q	Q	A	Q	G
11	Q	G	A	Q	A	G	Q	A	Q	G	A	Q	A	G	Q	A
12	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	G
13	G	Q	Q	Q	Q	Q	G	A	G	Q	Q	Q	Q	Q	G	A
14	Q	G	Q	Q	Q	G	Q	A	Q	G	Q	Q	Q	G	Q	A
15	Q	A	G	Q	G	A	Q	G	Q	A	G	Q	G	A	Q	G
16	Q	Q	Q	A	Q	Q	Q	A	Q	Q	Q	A	Q	Q	Q	A
17	A	Q	A	G	A	Q	A	G	A	Q	A	G	A	Q	A	G
18	Q	Q	Q	G	Q	Q	Q	G	Q	Q	Q	G	Q	Q	Q	G
19	G	Q	G	A	G	Q	G	A	G	Q	G	A	G	Q	G	A
20	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A	Q	A
21	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G
22	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
23	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Solution 12

Fig. 5. Transition tables and snapshots on 16 cells for the Solutions 9, 10, 11, and 12.

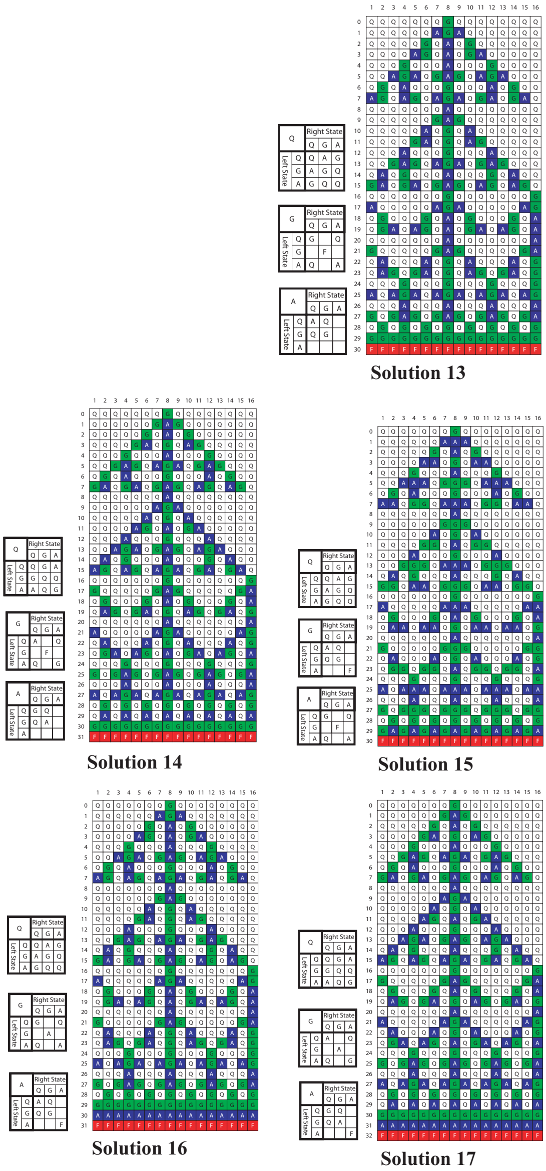


Fig. 6. Transition tables and snapshots on 16 cells for the Solutions 13, 14, 15, 16, and 17.

Q		Right State			
		Q	G	A	*
Left State	Q	Q	G	A	Q
	G	G	A	Q	A
	A	A	Q	G	
	*				

G		Right State			
		Q	G	A	*
Left State	Q	G	Q		G
	G	Q	A		A
	A			F	F
	*	G	A	F	

A		Right State			
		Q	G	A	*
Left State	Q	A		Q	A
	G		F		F
	A	Q		G	G
	*	A	F	G	

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	G	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
1	G	G	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
2	A	Q	G	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
3	A	Q	G	G	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
4	A	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
5	A	A	Q	G	G	G	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
6	G	Q	Q	Q	A	Q	G	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
7	G	G	Q	A	A	Q	G	G	Q	Q	Q	Q	Q	Q	Q	Q	Q
8	A	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	Q
9	A	A	Q	Q	Q	Q	Q	G	G	G	Q	Q	Q	Q	Q	Q	Q
10	G	Q	A	Q	Q	Q	G	Q	A	Q	G	Q	Q	Q	Q	Q	Q
11	G	Q	A	A	Q	G	G	Q	A	Q	G	G	Q	Q	Q	Q	Q
12	G	Q	Q	Q	Q	Q	Q	Q	A	Q	Q	Q	G	Q	Q	Q	Q
13	G	G	Q	Q	Q	Q	Q	A	A	A	Q	G	G	G	Q	Q	Q
14	A	Q	G	Q	Q	Q	A	Q	G	Q	Q	Q	A	Q	G	Q	Q
15	A	Q	G	G	Q	A	A	Q	G	G	Q	A	A	Q	G	G	Q
16	A	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	A
17	A	A	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	A	A
18	G	Q	A	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	A	Q	G
19	G	Q	A	A	Q	Q	Q	Q	Q	Q	Q	Q	Q	A	A	Q	G
20	G	Q	Q	Q	A	Q	Q	Q	Q	Q	Q	Q	A	Q	Q	Q	G
21	G	G	Q	A	A	A	Q	Q	Q	Q	Q	A	A	A	Q	G	G
22	A	Q	Q	Q	G	Q	A	Q	Q	Q	A	Q	G	Q	Q	Q	A
23	A	A	Q	G	G	Q	A	A	Q	A	A	Q	G	G	Q	A	A
24	G	Q	Q	Q	Q	Q	Q	Q	G	Q	Q	Q	Q	Q	Q	Q	G
25	G	G	Q	Q	Q	Q	Q	G	G	G	Q	Q	Q	Q	Q	G	G
26	A	Q	G	Q	Q	Q	G	Q	A	Q	G	Q	Q	Q	G	Q	A
27	A	Q	G	G	Q	G	G	Q	A	Q	G	G	Q	G	G	Q	A
28	A	Q	Q	Q	A	Q	Q	Q	A	Q	Q	Q	A	Q	Q	Q	A
29	A	A	Q	A	A	A	Q	A	A	A	Q	A	A	A	Q	A	A
30	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G	Q	G
31	G	A	G	A	G	A	G	A	G	A	G	A	G	A	G	A	G
32	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Fig. 7. Transition tables and snapshots on 17 cells for the converted solution.

Table 3. Time complexity and number of transition rules for Solutions 9, 10, 11, 12, 13, 14, 15, 16, and 17.

Solutions	Time complexity	# of transition rules	Notes
Solution 9	$T_{9,G}(n) = 3n/2, \quad T_{9,A}(n) = 2n$	19	
Solution 10	$T_{10,G}(n) = \begin{cases} 3 & k = 1 \\ 3n/2 - 2 & k \geq 2 \end{cases} \quad T_{10,A}(n) = \begin{cases} 2 & k = 1 \\ 2n - 2 & k \geq 2 \end{cases}$	19	$n = 2^k$
Solution 11	$T_{11,G}(n) = \begin{cases} 4 & k = 1 \\ 3n/2 & k \geq 2 \end{cases}$	25	$n = 2^k$
Solution 12	$T_{12,G}(n) = 3n/2 - 1, \quad T_{12,A}(n) = 2n - 1$	18	
Solution 13	$T_{13,G}(n) = \begin{cases} 2 & k = 1 \\ 2n - 2 & k \geq 2 \end{cases} \quad T_{13,A}(n) = 3n/2 - 2, k \geq 2$	18	$n = 2^k$
Solution 14	$T_{14,G}(n) = 2n - 1, \quad T_{14,A}(n) = 3n/2 - 1, k \geq 1$	18	
Solution 15	$T_{15,G}(n) = \begin{cases} 2 & k = 1 \\ 2n - 2 & k \geq 2 \end{cases} \quad T_{15,G}(n) = \begin{cases} 3 & k = 1 \\ 3n/2 - 2 & k \geq 2 \end{cases}$	19	$n = 2^k$
Solution 16	$T_{16,G}(n) = 2n - 1, \quad T_{16,A}(n) = 3n/2 - 1, k \geq 2$	19	
Solution 17	$T_{17,G}(n) = 2n, T_{17,A}(n) = 3n/2$	19	

4.4 Converting ring solutions into open-ring solutions

It is noted that most of the solutions presented above can be converted into the solutions for open-ring, that is, conventional one-dimensional array with the general at one end, without introducing any additional state. For example, Fig. 7 shows the transition rules and snapshots on 17 cells for a converted solution operating in optimum-steps. The solution can be obtained from the Solution 1 by adding 14 rules shown in Fig. 7 (left) illustrated with shaded small squares. It is noted that the solution obtained is also symmetric and optimum one in time.

5 Conclusion

An existence or non-existence of five-state firing squad synchronization protocol has been an outstanding and famous open problem for a long time. Å@In this paper, we answer partially to this problem by proposing a family of smallest four-state firing squad synchronization protocols that can synchronize any one-dimensional ring cellular array of length $n = 2^k$ for any positive integer k. The number $four$ is the smallest one known at present in the class of synchronization protocols proposed so far.

References

[1] R. Balzer. (1967): An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, vol. 10, pp. 22-42.

[2] A. Berthiaume, T. Bittner, L. Perković, A. Settle and J. Simon. (2004): Bounding the firing synchronization problem on a ring. *Theoretical Computer Science*, 320, pp. 213-228.

[3] Hans-D. Gerken: Über Synchronisations - Probleme bei Zellularautomaten. *Diplomarbeit*, Institut für Theoretische Informatik, Technische Universität Braunschweig, pp. 50.

[4] E. Goto. (1962): A minimal time solution of the firing squad problem. *Dittoed course notes for Applied Mathematics* 298, Harvard University, pp. 52-59, with an illustration in color.

[5] J. Mazoyer. (1987): A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, vol. 50, pp. 183-238.

[6] M. Minsky. (1967): *Computation: Finite and infinite machines.* Prentice Hall, pp. 28-29.

[7] E. F. Moore. (1964): The firing squad synchronization problem. in *Sequential Machines, Selected Papers* (E. F. Moore, ed.), Addison-Wesley, Reading MA., pp. 213-214.

[8] P. Sanders. (1994): Massively parallel search for transition-tables of polyautomata. In *Proc. of the VI International Workshop on Parallel Processing by Cellular Automata and Arrays*, (C. Jesshope, V. Jossifov and W. Wilhelmi (editors)), Akademie, 99-108.

[9] H. Umeo and N. Kamikawa. (2007): Four-state solutions to the firing squad synchronization problem based on Wolfram's rule 150. (a draft)

[10] H. Umeo and T. Yanagihara (2007): A smallest five-state solution to the firing squad synchronization problem. *Proc. of 5th Intern. Conf. on Machines, Computations, and Universality, MCU 2007*, LNCS 4664, pp.292-302.

[11] H. Umeo, J. B. Yunès, and N. Kamikawa (2008): About 4-state solutions to the firing squad synchronization problem. submitted to ACRI 2008.

[12] A. Waksman. (1966): An optimum solution to the firing squad synchronization problem. *Information and Control*, vol. 9 (1966), pp. 66-78.

[13] J. B. Yunès (2008): A 4-states algebraic solution to linear cellular automata synchronization. *International Processing Letters*, (to appear).

New approach to search for gliders in cellular automata

Emmanuel Sapin

Faculty of Computing, Engineering and Mathematical Sciences,
University of the West of England, Bristol BS16 1QY, UK
emmanuelsapin@hotmail.com

Abstract. This paper deals with a new method to search for mobile self-localized patterns of non-resting states in cellular automata. The spontaneous emergence of such patterns called gliders have been searched throughout all the automata of a specific space. As a result, thousands of novel gliders were discovered through the subsequent experiments and studied in terms of speed and period thanks to an automatic process of identification of gliders.

1 Introduction

The emergence of computation in complex systems with simple components is a hot topic in the science of complexity [23]. A uniform framework to study this emergent computation in complex systems is cellular automata [15]. They are discrete systems in which an array of cells evolve from generation to generation on the basis of local transition rules [22].

The well-established problems of emergent computation and universality in cellular automata has been tackled by a number of people in the last thirty years [4, 12, 10, 14, 1] and remains an area where amazing phenomena at the edge of theoretical computer science and non-linear science can be discovered.

The most well-known universal automaton is the Game of Life [7]. It was shown to be universal by Conway [5] who employed *gliders*. The latter are mobile self-localized patterns of non-resting states, used by Conway to carry information. Such gliders are the basic element for showing the universality in Conway's demonstration.

The search for gliders is then a new study of science of complexity and was notably explored by Adamatzky et al. with a phenomenological search [13], Wuensche, who used his Z-parameter and entropy [25] and Eppstein [6]. Lohn et al. [11] and Ventrella [21] have searched for gliders using genetic algorithms [8], while, Sapin et al. [18, 19] used genetic algorithms to evolve transition rules of cellular automata then, searched for gliders exhibited by the evolved automata.

In this work, a new method of research for gliders is presented. Instead of using any method to chose transition rules of cellular automata, the challenge here is to search for gliders in all the automata of a space. As the ones use by adamtzky et al. [2] as models of reaction-diffusion excitable chemical systems, ternary totalistic cellular automata are chosen. Then, in order to limited the search space, cellular automata using the four nearest neigbours to updates their cell states are considered. Following on from this, the space called $\mathcal{V}$, of 2D ternary totalistic cellular automata using the four nearest neigbours to updates their cell states at the next generation are chosen.

The first section presentes notations and definitions, the second describes the search for gliders, while those which are found are described in the next section. Finally, the last section summarizes the presented results and discusses directions for future research.

2 Definitions and notations

In this section, some definitions and notations about cellular automata are presented.

2.1 Cellular automata

A local transition rule of a 2D ternary cellular automata using the four nearest neighbours to updates their cell states at the next generation is a function ϕ : {0, 1, 2}^{5}→ {0, 1, 2}. Among these automata, *totalistic* automata are studied. An automaton is totalistic iff a cell-state is updated depending on just the numbers, not positions of different cell-states in its neighbourhoods. The functions of the local transition rule are then:

$$\phi(q_1, q_2, q_3, q_4, q_5) = f(\sigma 1(x)^t, \sigma 2(x)^t) \tag{1}$$

where f : {0, . . . , 5}^{2}→{0, 1, 2} and $\sigma p(x)^t$ is the number of cell x's neighbours with cell-state p∈{0, 1, 2} at time step t.

To give a compact representation of the cell-state transition rule, the formalism in [3] is adopted. The cell-state transition rule is represented as a matrix M = (M_{ij}), where $0 \leq i \leq j \leq 7$, $0 \leq i + j \leq 5$, and M_{ij}∈{0, 1, 2}. The output state of each neighbourhood is given by the row-index i (the number of neighbours in cell-state 1) and column-index j (the number of neighbours in cell-state 2). We do not have to count the number of neighbours in cell-state 0, because it is given by 7 - (i + j). The matrix M contains then 21 elements.

Ternary automata are considered, in which the state 0 is a *dedicated substrate state*. A state 0 is dedicated substrate state iff a cell in state 0, whose neighbourhood is filled only with states 0, does not change its state, 0 is analogue of quiescent state in cellular automaton models [24]. Then M_{00} is equal 0 in all automata so the studied space $\mathcal{V}$ contains 3^{20}.

A glider, called G and exhibited by some automata of $\mathcal{V}$, is presented generation after generation as the one from figure 1 while the set of automata of $\mathcal{V}$ that exhibit this glider is determined by the process used in [20].

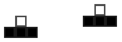


Fig. 1. Glider of period 1 exhibited by some automata of $\mathcal{V}$ shown generation after generation along its period.

$$\begin{pmatrix} 0 & 1 & X & X & X & X \\ 0 & 2 & X & X & X & \\ 0 & X & X & X & & \\ 0 & 0 & X & & & \\ X & X & & & & \\ X & & & & & \end{pmatrix}$$

Table 1. Transition rule of automata that exhibits the glider G.

Finally, the elements of M that can vary without changing the evolution of G is represented by the letter X on table 1, that shows the transition rules of the automata that exhibit G.

In turn, fourteen elements of M can change their value without changing the evolution of G so 3^{14} of the 3^{20} automata of $\mathcal{V}$ exhibit the glider G.

3 Search for gliders

3.1 Search method

Gliders are searched for in all automata of $\mathcal{V}$. The search method is inspired by the one use in [17]. A random configuration of cells is generated in a square of 40×40 centered in a 200×200 space. These cells evolve during three hundred generations with the transition rule of the tested automaton. At each generation, each group of connected cells, figure 2, is isolated in an empty space and evolves during 30 generations. For each generation, the original pattern is searched in the test universe. Three situations can arise:

- The initial pattern has reappeared at its first location, it is then periodic and the simulation do not need to continue.
- It has reappeared at another location, it is then a glider and the current generation is the period of the glider.
- It has not reappeared.

3.2 Number of gliders

In order to determine how many different gliders were found, an automatic system that determines if a glider is new is required. At least two criteria can be used:

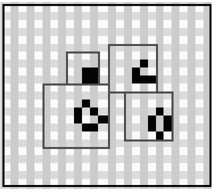


Fig. 2. Groups of isolated cells.

- Shapes of gliders.
- Sets of used neighbourhood states.

The shapes of gliders are not taken into account because the gliders can be found in different orientations and in different generations. So, in order to determine if a glider is new, the set of neighbourhood states used by the given gun are compared to the ones of the other guns. For each gun and each neighbourhood state, four cases exist:

- The neighbourhood state is not used by this gun.
- The neighbourhood state is used by this gun and the value of the central cell at the next generation is 0, 1 or 2.

Two gliders are different iff at least one neighbourhood state is not in the same case for the two gliders.

Thanks to this qualification of different gliders through the experimentations, 5186 different ones were discovered, all of them emerging spontaneously from random configurations of cells. The 5186 gliders can be found in [16] in mcl format.

4 Description of gliders

The period and the speed of the found gliders are described in the first subsection below then some specific gliders are shown in the last part.

4.1 Characteristic of the gliders

The distribution of the periods of the found gliders is shown figure 3.

There are very few gliders of period 1 and no gliders beyond period 4 were found.

The distribution of the speeds of the found gliders is shown in figure 4. In order to ascertain this, the position after a period is compared to the initial position, with the speed being in cells per generation and only the four direct neighbours of a cell C are spaces by 1 while the diagonal cells are spaced by 2.

All the gliders with speeds of $\frac{1}{3}$ and $\frac{2}{3}$ have a period of 3 and the other gliders of period 3 have a speed of 1. Only the gliders of period 4 are capable of a speed of $\frac{1}{4}$,and finally, the gliders of period 1 have a speed of 1.

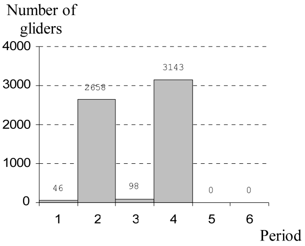


Fig. 3. Distribution of the periods of the found gliders.

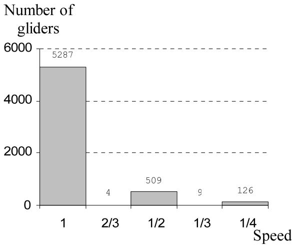


Fig. 4. Distribution of the speeds of the found gliders.

3267 gliders move horizontally or vertically and the others move along diagonals.

4.2 Specific gliders

The glider accepts by the highest number of automata is the one in Fig. 1. It is also one with the highest speed of 1.

A glider with the slowest speed of $\frac{1}{4}$ is shown in Fig. 5, and is exhibited by the automata of Tab. 2.

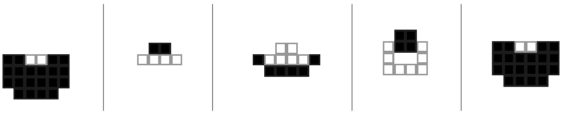


Fig. 5. Glider of period 4.

This glider has a period of 4 and moves by 1 cell per period.

A period 2 diagonal glider of speed 1, exhibited by automata of Tab. 3, is shown Fig. 6.

$$\begin{pmatrix} 0 & 2 & 1 & 2 & X & X \\ 0 & 0 & 2 & 1 & 0 & \\ 0 & 1 & 1 & X & & \\ 0 & 1 & X & & & \\ 0 & 0 & & & & \\ 0 & & & & & \end{pmatrix}$$

Table 2. Transition rule of automata that exhibits the glider shown in Fig. 5

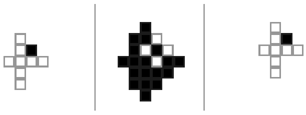


Fig. 6. Diagonal glider.

$$\begin{pmatrix} 0 & 2 & 2 & 2 & X & 2 \\ 0 & 1 & 2 & 1 & 1 & \\ 0 & 1 & X & X & & \\ 0 & 0 & 0 & & & \\ 0 & 1 & & & & \\ 0 & & & & & \end{pmatrix}$$

Table 3. Transition rule of automata that exhibits the glider shown in Fig. 6.

5 Synthesis and perspectives

This paper deals with the emergence of computation in complex systems with local interactions, while more particularly, this paper presents a new approach to searching for gliders in cellular automata.

The presented search method was to look for the appearance of gliders from the evolution of random configuration of cells for all automata of a space. 2D ternary totalistic automata using the four nearest neighbours to updates their cell states were chosen, 5186 gliders exhibited by automata of this space were found. These gliders were not known before and they have been studied in terms of speed and period.

Further goals can be to find glider guns by the same method in this space of automata or to extend the method to other spaces of automata. All these automata may be potential candidates for being shown as universal automata and future work could be aimed at proving the universality of some of them.

Future work could also be to calculate for each automata some rule-based parameters, e.g., Langton's λ [9]. All automata exhibiting gliders may have similar values for these parameters, which could lead to a better understanding of the link between the rule transition and the emergence of computation in cellular automata, and therefore, the emergence of computation in complex systems with simple components.

6 Acknowledgements

The work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of the United Kingdom, Grant EP/E005241/1.

References

[1] A. Adamatzky. Universal dymical computation in multi-dimensional excitable lattices. *International Journal of Theoretical Physics*, 37:3069–3108, 1998.

[2] A. Adamatzky and A. Wuensche. Computing in spiral rule reaction-diffusion hexagonal cellular automaton. *Complex Systems*, 16, 2007.

[3] A. Adamatzky, A. Wuensche, and B. De Lacy Costello. Glider-based computing in reaction-diffusion hexagonal cellular automata. *Chaos, Solitons and Fractals 27*, pages 287–295, 2006.

[4] E. R. Banks. *Information and transmission in cellular automata.* PhD thesis, MIT, 1971.

[5] E. Berlekamp, J. H. Conway, and R. Guy. Winning ways for your mathematical plays. *Academic Press, New York*, 1982.

[6] D. Eppstein. `http://www.ics.uci.edu/~eppstein/ca/`.

[7] M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.

[8] J. H. Holland. Adaptation in natural and artificial systems. *University of Michigan*, 1975.

[9] C. L. Langton. Computation at the edge of chaos. *Physica D*, 42, 1990.

[10] K. Lindgren and M. Nordahl. Universal computation in simple one dimensional cellular automata. *Complex Systems*, 4:299–318, 1990.

[11] J.D. Lohn and J.A. Reggia. Automatic discovery of self-replicating structures in cellular automata. *IEEE Transactions on Evolutionary Computation*, 1:165–178, 1997.

[12] N. Margolus. Physics-like models of computation. *Physica D*, 10:81–95, 1984.

[13] G. J. Martinez, A. Adamatzky, and H. V. McIntosh. Phenomenology of glider collisions in cellular automaton rule 54 and associated logical gates chaos. *Fractals and Solitons*, 28:100–111, 2006.

[14] K. Morita, Y. Tojima, I. Katsunobo, and T. Ogiro. Universal computing in reversible and number-conserving two-dimensional cellular spaces. *In A. Adamatzky (ed.), Collision-Based Computing, Springer Verlag.*, pages 161–199, 2002.

[15] J. Von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana, Ill., 1966.

[16] E. Sapin. http://uncomp.uwe.ac.uk/sapin/glider.zip, 2007.

[17] E. Sapin, O. Bailleux, and J.J. Chabrier. Research of a cellular automaton simulating logic gates by evolutionary algorithms. *EuroGP03. Lecture Notes in Computer Science*, 2610:414–423, 2003.

[18] E. Sapin, O. Bailleux, J.J. Chabrier, and P. Collet. A new universel automata discovered by evolutionary algorithms. *Gecco2004.Lecture Notes in Computer Science*, 3102:175–187, 2004.

[19] E. Sapin, O. Bailleux, J.J. Chabrier, and P. Collet. Demonstration of the universality of a new cellular automaton. *IJUC*, 2(3), 2006.

[20] E. Sapin, L. Bull, and A. Adamatzky. A genetic algorithm approach to searching for glider guns in cellular automata. *IEEE.Lecture Notes in Computer Science.* In press.

[21] J.J. Ventrella. A particle swarm selects for evolution of gliders in non-uniform 2d cellular automata. *In Artificial Life X: Proceedings of the 10th International Conference on the Simulation and Synthesis of Living Systems*, pages 386–392, 2006.

[22] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[23] S. Wolfram. *A New Kind of Science*. Wolfram Media, Inc., Illinois, USA, 2002.

[24] S. Wolfram and N.H. Packard. Two-dimensional cellular automata. *Journal of Statistical Physics*, 38:901–946, 1985.

[25] A. Wuensche. Discrete dinamics lab (ddlab),www.ddlab.org, 2005.

Chaotic properties of elementary cellular automaton rule 168 of Wolfram class I

Fumio Ohi

Nagoya Institute of Technology,
Gokiso-cho, Showa-ku, Nagoya, 466-8555, Japan
ohi.fumio@nitech.ac.jp

Abstract. This work examines the chaotic properties of the elementary cellular automaton rule 168 of Wolfram class I.
Wolfram's classification based on a computer simulation is well known and the time-space patterns generated by the members of the class I is known to die out in a finite time and these cellular automata might not be attractive at first glance.
F.Ohi [4] has shown that a dynamical system defined by rule 40 of Wolfram's class I is Devaney chaos on the class of configurations of specific patterns and an exact calculation of the spreading rates of the specific configurations were shown at Automata 2007 in Toronto, where we showed that for every α such that $1/2 \leq \alpha \leq 1$, there exists a configuration of which right spreading rate is α. The spreading rate is equivalent to Lyapunov exponent defined by M. A. Shereshevsky [5] for rule 40.
In this work we show that rule 168, which is also a member of Wolfram's class I and in one sense contains rule 40, defines a chaotic dynamical system on a class of configurations of specific type and show that for every α such that $0 < \alpha \leq 1$, there exists a configuration of which right spreading rate is α, for the proof of which we use Ergodic theory for an interval dynamical system.

1 Introduction

Wolfram's classification based on an extensive computer simulation is well known and the time-space patterns generated by the members of the class I is known to die out in a finite time and might not be attractive at first glance.

F.Ohi [4] has shown that a dynamical system defined by the rule 40 of Wolfram class I is Devaney chaos on the class of configurations having specific patterns and an exact calculation of the spreading rates of the specific configurations were shown at Automata 2007 in Toronto, where we had that for every α such that $1/2 \leq \alpha \leq 1$, there exists a configuration of which spreading rate is α. The spreading rate is equivalent to Lyapunov exponent defined by M. A. Shereshevsky [5] for the case of rule 40.

In this paper we examine the dynamical system defined by rule 168, which is also a member of Wolfram's class I, and show that it has a chaotic sub-dynamical system and also, using Ergodic theorem, for every α such that $0 < \alpha \leq 1$, there exists a configuration of which spreading rate is α. Furthermore, rule 168 is shown to contain in one sense rule 40 as a sub-dynamical system.

An elementary cellular automaton (ECA) is defined to be a tuple $(\{0,1\}, g)$, where g is a mapping from $\{0,1\}^3$ to $\{0,1\}$ and is called a local transition function. An ECA is determined by g and is then we simply call it an ECA g. There exist $2^8 = 256$ ECAs and each of them has the rule number defined by $\sum_{a,b,c} g(a,b,c)2^{a2^2+b2+c}$. We write the local transition function having rule number r as g_r.

The local transition functions g_{40}, g_{168} are given by the following table.

(a,b,c)	$(1,1,1)$	$(1,1,0)$	$(1,0,1)$	$(1,0,0)$	$(0,1,1)$	$(0,1,0)$	$(0,0,1)$	$(0,0,0)$
$g_{40}(a,b,c)$	0	0	1	0	1	0	0	0
$g_{168}(a,b,c)$	1	0	1	0	1	0	0	0

We hardly imagine that these rules have chaotic properties by observing the time-space patterns generated by computer simulations for randomly given initial configurations. However, these cellular automata have chaotic properties and rule 168 contains in one sense the chaotic properties of rule 40, which are explained in the following sections.

An ECA g defines a mapping $\boldsymbol{g}$ from $\mathcal{A} \equiv \{0,1\}^{\mathbf{Z}}$ to $\mathcal{A}$, which is called the global transition function of the ECA, as

$$\boldsymbol{x} = (\cdots, x_{-1}, x_0, x_1, \cdots) \in \mathcal{A}, \quad (\boldsymbol{g}(\boldsymbol{x}))_i = g(x_{i-1}, x_i, x_{i+1}), \quad i \in \boldsymbol{Z}.$$

We usually use the bold face letter of the letter denoting the local transition function to show the corresponding global transition function. An element of $\mathcal{A}$ is called a configuration.

The left shift and right shift transformations are written as $\sigma_L : \mathcal{A} \to \mathcal{A}$ and $\sigma_R : \mathcal{A} \to \mathcal{A}$, respectively.

Defining a metric d on $\mathcal{A}$ as

$$\boldsymbol{x},\ \boldsymbol{y} \in \mathcal{A}, \quad d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=-\infty}^{\infty} \frac{|x_i - y_i|}{2^{|i|}},$$

we have a topological dynamical system $(\mathcal{A}, \boldsymbol{g})$, which defines an orbit for an arbitrarily given initial configuration $\boldsymbol{x} \in \mathcal{A}$ as

$$\boldsymbol{g}^0(\boldsymbol{x}) = \boldsymbol{x}, \quad \boldsymbol{g}^{t+1}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{g}^t(\boldsymbol{x})),\ t \geq 0.$$

The time-space pattern generated by $\boldsymbol{g}$ with an initial configuration $\boldsymbol{x} \in \mathcal{A}$ is the set $\{(t, \boldsymbol{g}^t(\boldsymbol{x})), t \geq 0\}$. Our general problem is to analyze the dynamical system and characterize the time-space pattern.

A topological dynamical system $(\mathcal{S}, \boldsymbol{g})$ is a sub-dynamical system of $(\mathcal{A}, \boldsymbol{g})$ if $\mathcal{S} \subseteq \mathcal{A}$ and $\boldsymbol{g}(\mathcal{S}) \subseteq \mathcal{S}$. The metric on $\mathcal{S}$ is the one defined on $\mathcal{A}$.

A topological dynamical system $(\mathcal{S}, \boldsymbol{g})$ is called Devaney chaos when it has a dense orbit and the class of all periodic configuration is dense in $\mathcal{S}$. (See G. Cattaneo, et al. [2].)

In this paper we use the following notations for our rigorous examination of time-space patterns generated by rule 168.

1.1 Notations

(1) For $\boldsymbol{\alpha}_i \in \{0,1\}^{n_i},\ \boldsymbol{\beta}_i \in \{0,1\}^{m_i},\ n_i \geq 1,\ m_i \geq 1,\ i \in \mathbf{Z}$, we define

$$\begin{aligned}(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)_{i=-\infty}^{+\infty} =& (\cdots, \alpha_1^{-1}, \cdots, \alpha_{n-1}^{-1}, \beta_1^{-1}, \cdots, \beta_{m-1}^{-1}, \\ & \alpha_1^0, \cdots, \alpha_{n_0}^0, \beta_1^0, \cdots, \beta_{m_0}^0, \alpha_1^1, \cdots, \alpha_{n_1}^1, \beta_1^1, \cdots, \beta_{m_1}^1, \cdots),\end{aligned}$$

where $\boldsymbol{\alpha}_i = (\alpha_1^i, \cdots, \alpha_{n_i}^i),\ \boldsymbol{\beta}_i = (\beta_1^i, \cdots, \beta_{m_i}^i),\ i \in \mathbf{Z}$.
(2) $\mathbf{0}$ means one of the three types $(\cdots, 0, 0, 0, \cdots),\ (\cdots, 0, 0,)$ and $(0, 0, \cdots)$. It is clear from the context which type $\mathbf{0}$ means. We also use the terminology $\mathbf{0}_n = (\underbrace{0, \cdots, 0}_{n}),\ n \in \mathbf{N}$, where $\mathbf{N}$ is the set of nonnegative integers. 1 and $\mathbf{1}_n$ are interpreted similarly to 0 and $\mathbf{0}_n$, respectively. When $n = 0$, the blocks are empty.
(3) We intensively use the following terminology.

$$\begin{aligned}\mathcal{S}_{0(m_1,\cdots,m_p),1(n_1,\cdots,n_q)} &= \{(\mathbf{0}_{i_j}, \mathbf{1}_{k_j})_{j=-\infty}^{\infty} \mid i_j = m_1 \text{ or} \cdots \text{or } m_p, \\ &\qquad\qquad k_j = n_1 \text{ or} \cdots \text{or } n_q\}\ , \\ \mathcal{S}_{0(\leq m),1(\leq n)} &= \{(\mathbf{0}_{i_j}, \mathbf{1}_{k_j})_{j=-\infty}^{\infty} \mid 1 \leq i_j \leq m, 1 \leq k_j \leq n\}\ , \\ \mathcal{S}_{0(\leq m)} &= \cup_{n=1}^{\infty} \mathcal{S}_{0(\leq m),1(\leq n)}\ , \\ \mathcal{S}_{1(\leq n)} &= \cup_{m=1}^{\infty} \mathcal{S}_{0(\leq m),1(\leq n)}\ .\end{aligned}$$

(4) For $\boldsymbol{x} \in \mathcal{A}$, $\boldsymbol{x}_{i,\to} = (x_i, x_{i+1}, \cdots), \boldsymbol{x}_{\leftarrow,i}(\cdots, x_{i-1}, x_i), \boldsymbol{x}_{i,j} = (x_i, \cdots, x_j)$ $(i < j)$, and for a subset $\mathcal{S}$ of $\mathcal{A}$,

$$\mathcal{S}^+ = \{\boldsymbol{x}_{0,\to} \mid \boldsymbol{x} \in \mathcal{S}\ \}\ .$$

2 Spreading rate and Lyapunov exponent

Following Shereshevsky [5], Lyapunov exponent of $\boldsymbol{x} \in \mathcal{A}$ is defined by the following procedure. For $s \in \mathbf{Z}$, let

$$\begin{aligned}W_s^+(\boldsymbol{x}) &\equiv \{\boldsymbol{y} \in \mathcal{A} \mid \forall i \geq s,\ y_i = x_i\}\,, \\ W_s^-(\boldsymbol{x}) &\equiv \{\boldsymbol{y} \in \mathcal{A} \mid \forall i \leq s,\ y_i = x_i\}\,, \\ \tilde{\Lambda}_t^+(\boldsymbol{x}) &\equiv \min\left\{s \mid \boldsymbol{g}^t(W_0^+(\boldsymbol{x})) \subset W_s^+(\boldsymbol{g}^t(\boldsymbol{x}))\right\}, \\ \Lambda_t^+(\boldsymbol{x}) &\equiv \max_{j \in \mathbf{Z}} \left\{\tilde{\Lambda}_t^+(\sigma_L^j \boldsymbol{x})\right\}, \\ \tilde{\Lambda}_t^-(\boldsymbol{x}) &\equiv \max\left\{s \mid \boldsymbol{g}^t(W_0^-(\boldsymbol{x})) \subset W_s^-(\boldsymbol{g}^t(\boldsymbol{x}))\right\}, \\ \Lambda_t^-(\boldsymbol{x}) &\equiv \min_{j \in \mathbf{Z}} \left\{\tilde{\Lambda}_t^-(\sigma_L^j \boldsymbol{x})\right\},\end{aligned}$$

where $\sigma_L^j = \sigma_R^{-j}$ for $j < 0$. When

$$\lim_{t\to\infty} \frac{\Lambda_t^+(\boldsymbol{x})}{t} \quad \text{and} \quad \lim_{t\to\infty} \frac{\Lambda_t^-(\boldsymbol{x})}{t}$$

exist, we call them right and left Lyapunov exponents of $\boldsymbol{x}$, respectively.

For $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{A}$, we set

$$DFR(\boldsymbol{x}, \boldsymbol{y}) \equiv \sup\{i \mid x_i \neq y_i\} \quad \text{and} \quad DFL(\boldsymbol{x}, \boldsymbol{y}) \equiv \inf\{i \mid x_i \neq y_i\}.$$

The next Lemma combines the spreading rate and Lyapunov exponent.

Lemma 2.1. *For ECA g, configuration $\boldsymbol{x} \in \mathcal{A}$ and $t \in \mathbf{N}_+$, we have*

$$\begin{aligned}
(1) \quad & \max_{\boldsymbol{y} \in W_0^+(\boldsymbol{x})} DFR(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(\boldsymbol{y})) \\
& \qquad = \min\left\{ \; s \mid \boldsymbol{g}^t(W_0^+(\boldsymbol{x})) \subset W_s^+(\boldsymbol{g}^t(\boldsymbol{x}))\right\} - 1, \\
(2) \quad & \min_{\boldsymbol{y} \in W_0^-(\boldsymbol{x})} DFL(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(\boldsymbol{y})) \\
& \qquad = \max\left\{s \mid \boldsymbol{g}^t(W_0^-(\boldsymbol{x})) \subset W_s^-(\boldsymbol{g}^t(\boldsymbol{x}))\right\} + 1.
\end{aligned}$$

Following Ilachinski [3], the spreading rate of a configuration $\boldsymbol{x} \in \mathcal{A}$ is defined as the following. First we set $n_j : \mathcal{A} \to \mathcal{A}$ $(j \in \mathbf{Z})$ as

$$\boldsymbol{x} \in \mathcal{A}, \quad (n_j(\boldsymbol{x}))_i = \begin{cases} x_i, \, i \neq j, \\ \overline{x_i}, \, i = j. \end{cases}$$

n_j reverses the state of the j-th site of the configuration $\boldsymbol{x}$. For ECA g, letting

$$\begin{aligned}
\Gamma_t^+(\boldsymbol{x}) &\equiv \max_{j \in \mathbf{Z}} \left\{DFR(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(n_j(\boldsymbol{x})) - j\right\} \;, \\
\Gamma_t^-(\boldsymbol{x}) &\equiv \min_{j \in \mathbf{Z}} \left\{DFL(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(n_j(\boldsymbol{x})) - j\right\} \;,
\end{aligned}$$

when

$$\gamma^+(\boldsymbol{x}) = \lim_{t \to \infty} \Gamma_t^+(\boldsymbol{x})/t \quad \text{and} \quad \gamma^-(\boldsymbol{x}) = \lim_{t \to \infty} \Gamma_t^-(\boldsymbol{x})/t$$

exist, we call them right and left spreading rates of $\boldsymbol{x}$, respectively.

Theorem 2.1. *For ECA g, at $\boldsymbol{x} \in \mathcal{A}$ and $t \in \mathbf{N}$, we have*

$$\begin{aligned}
(1) \quad & \Lambda_t^+(\boldsymbol{x}) = \max_{j \in \mathbf{Z}} \max_{\boldsymbol{y} \in W_j^+(\boldsymbol{x})} \left\{DFR(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(\boldsymbol{y})) - j\right\} + 1, \\
& \Gamma_t^+(\boldsymbol{x}) \leq \Lambda_t^+(\boldsymbol{x}). \\
(2) \quad & \Lambda_t^-(\boldsymbol{x}) = \min_{j \in \mathbf{Z}} \min_{\boldsymbol{y} \in W_j^-(\boldsymbol{x})} \left\{DFR(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(\boldsymbol{y})) - j\right\} - 1, \\
& \Lambda_t^-(\boldsymbol{x}) \leq \Gamma_t^-(\boldsymbol{x}).
\end{aligned}$$

Spreading rate does not generally coincide with Lyapunov exponent, but they are generally equivalent when ECA is left most permutive or right most permutive. Rule 40 and 168 are neither right nor left most permutive, but for the configurations which we are going to treat in this paper they are coincide with each other.

3 Chaotic property of rule 168

F. Ohi [4] shows that

$$\begin{aligned} \forall \boldsymbol{x} \in \mathcal{A}\backslash\mathcal{S}_{0(1),1(1,2)}, &\quad \lim_{t\to\infty} \boldsymbol{g}_{40}^t(\boldsymbol{x}) = \boldsymbol{0}, \\ \forall \boldsymbol{x} \in \mathcal{S}_{0(1),1(1,2)}, &\quad \boldsymbol{g}_{40}(\boldsymbol{x}) = \sigma_L(\boldsymbol{x}), \end{aligned}$$

and the sub-dynamical system $(\mathcal{S}_{0(1),1(1,2)}, \boldsymbol{g}_{40})$ is Devaney chaos.

In this section, examining the orbits of $\boldsymbol{x} \in \mathcal{A}$ by g_{168} and using the method of F. Ohi [4], we show that $\boldsymbol{g}_{168} = \sigma_L$ on $\mathcal{S}_{0(1),1(\leq m)}$ and then is Devaney chaos. It is also shown that the spreading rate of an element of $\mathcal{S}_{0(1),1(\leq m)}$ by $\boldsymbol{g}_{168}$ is equivalent to the relative frequency of some patterns relating to an invariant measure of $(\mathcal{S}^+_{0(1),1(\leq m)}, \sigma_L)$ and also related to an interval dynamical system $([0,1], f)$ which is defined to be conjugate to $(\mathcal{S}^+_{0(1),1(\leq m)}, \sigma_L)$

From the table of g_{168} in the section 1, the next two Lemmas are clear.

Lemma 3.1. *For a configuration $\boldsymbol{x} \in \mathcal{A}$, if $x_i = x_{i+1} = 0$, then*

$$(\boldsymbol{g}_{168}(\boldsymbol{x}))_{i,i+1} = (0,0) \ .$$

A block $(0,0)$ in the configuration $\boldsymbol{x}$ plays like a wall in the time development of the configuration by $\boldsymbol{g}_{168}$. More precisely we have the following formulas for $\boldsymbol{g}^t_{168}(\boldsymbol{x})$ at time step t.

$$\boldsymbol{g}^t_{168}(\boldsymbol{x}) = \left(\left(\boldsymbol{g}^t_{168}(\boldsymbol{x}_{\leftarrow,i-1}, \boldsymbol{0})\right)_{\leftarrow,i-1}, \overset{\underset{\smile}{i}}{0}, \overset{\underset{\smile}{i+1}}{0}, \left(\boldsymbol{g}^t_{168}(\boldsymbol{0}, \boldsymbol{x}_{i+2,\rightarrow})\right)_{i+2,\rightarrow} \right)$$

Lemma 3.2. *(1) Let $\boldsymbol{x} = (\cdots, \overset{\underset{\smile}{-1}}{0}, \overset{\underset{\smile}{0}}{0}, 0, \boldsymbol{1}_{m_1}, 0, \boldsymbol{1}_{m_2}, 0, \boldsymbol{1}_{m_3}, \cdots)$. The site numbers -1 and 0 of the block $(0,0)$ does not lose the generality. In this case we have*

$$\begin{aligned} \boldsymbol{g}^{m_1}_{168}(\boldsymbol{x}) &= (\cdots, \overset{\underset{\smile}{-1}}{0}, \overset{\underset{\smile}{0}}{0}, \overset{\underset{\smile}{1}}{0}, 0, \boldsymbol{1}_{m_2}, 0, \boldsymbol{1}_{m_3}, \cdots) \ , \\ \boldsymbol{g}^{m_1+\cdots+m_n}_{168}(\boldsymbol{x}) &= (\cdots, \overset{\underset{\smile}{-1}}{0}, \overset{\underset{\smile}{0}}{0}, \underbrace{0, \cdots, 0}_{n}, 0, \boldsymbol{1}_{m_{n+1}}, \cdots) \ . \end{aligned}$$

(2) For $\boldsymbol{x} = (\cdots, \overset{\underset{\smile}{-1}}{0}, \overset{\underset{\smile}{0}}{0}, 0, \boldsymbol{1}_{m_1}, 0, \boldsymbol{1}_{m_2}, \cdots 0, \boldsymbol{1}_{m_n}, \boldsymbol{0})$ we have

$$\begin{aligned} \boldsymbol{g}^{m_1}_{168}(\boldsymbol{x}) &= (\cdots, \overset{\underset{\smile}{-1}}{0}, \overset{\underset{\smile}{0}}{0}, \overset{\underset{\smile}{1}}{0}, 0, \boldsymbol{1}_{m_2}, \cdots 0, \boldsymbol{1}_{m_n}, \boldsymbol{0}) \ , \\ \boldsymbol{g}^{m_1+\cdots+m_n}_{168}(\boldsymbol{x}) &= (\cdots, \overset{\underset{\smile}{-1}}{0}, \overset{\underset{\smile}{0}}{0}, \underbrace{0, \cdots, 0}_{n}, \boldsymbol{0}) \ , \\ \left(\boldsymbol{g}^t_{168}(\boldsymbol{x})\right)_{-1,\rightarrow} &= \boldsymbol{0} \ , \quad \forall t \geq m_1 + \cdots + m_m \ . \end{aligned}$$

(3) For $\boldsymbol{x} = (\cdots, \overset{-1}{0}, \overset{0}{0}, 0, \mathbf{1}_{m_1}, 0, \mathbf{1}_{m_2}, \cdots 0, \mathbf{1}_{m_n}, 0, \mathbf{1})$ *we have*

$$\begin{aligned}
\boldsymbol{g}^{m_1}_{168}(\boldsymbol{x}) &= (\cdots, \overset{-1}{0}, \overset{0}{0}, \overset{1}{0}, 0, \mathbf{1}_{m_2}, \cdots 0, \mathbf{1}_{m_n}, 0, \mathbf{1}) \ , \\
\boldsymbol{g}^{m_1+\cdots+m_n}_{168}(\boldsymbol{x}) &= (\cdots, \overset{-1}{0}, \overset{0}{0}, \underbrace{0, \cdots, 0}_{n}, 0, \mathbf{1}) \ , \\
\big(\boldsymbol{g}^{t}_{168}(\boldsymbol{x})\big)_{-1,\rightarrow} &= (0, 0, \underbrace{0, \cdots, 0}_{n}, 0, \mathbf{1}) \ , \quad \forall t \geq m_1 + \cdots + m_m \ .
\end{aligned}$$

(4) For $\boldsymbol{x} = (\cdots, \underbrace{\overset{0}{0}, 0, \cdots, 0\ \overset{i}{0}}_{\geq 2}, \cdots)$ *we have*

$$\begin{aligned}
\boldsymbol{g}(\boldsymbol{x}) &= (\cdots, \overset{-1}{0}, \overset{0}{0}, 0, \cdots, 0, \overset{i}{0}, \cdots) \\
\boldsymbol{g}^{t}(\boldsymbol{x}) &= (\cdots, \overset{-t}{0}, 0, \cdots, 0, \overset{0}{0}, 0, \cdots, 0, \overset{i}{0}, \cdots) \ , \quad \forall t \geq 1 \ .
\end{aligned}$$

From Lemmas 3.1, 3.2 and the table of g_{168}, the next Theorem is easily given.

Theorem 3.1. *(1)* $\boldsymbol{x} \in \mathcal{S}_{0(1),1(\leq k)}$, $\boldsymbol{g}_{168}(\boldsymbol{x}) = \sigma_L(\boldsymbol{x})$.

(2) For $\boldsymbol{x} \in \mathcal{A} \backslash \mathcal{S}_{0(1)}$*;*

(2-i) If 0*-state sites of* $\boldsymbol{x}$ *are isolated and* $\exists i \in \mathbf{Z}$, $\boldsymbol{x}_{i,\rightarrow} = \mathbf{1}$*, then*

$$\forall t \geq 1 \ , \ \boldsymbol{g}^{t}_{168}(\boldsymbol{x}) \neq \mathbf{1} \ , \qquad \lim_{t\to\infty} \boldsymbol{g}^{t}_{168}(\boldsymbol{x}) = \mathbf{1} \ .$$

(2-ii) If $\boldsymbol{x} = \mathbf{1}$ *, then* $\boldsymbol{g}^{t}_{168}(\mathbf{1}) = \mathbf{1}$ *, and thus*

$$\forall t \geq 1 \ , \ \boldsymbol{g}^{t}_{168}(\mathbf{1}) = \mathbf{1} \ .$$

(2-iii) If $\exists i \in \mathbf{Z}$*,* $\boldsymbol{x}_{i,\rightarrow} = \mathbf{1}$, $\exists j \in \mathbf{Z}$*,* $(x_j, x_{j+1}) = (0, 0)$ *(the right most 00 block in* $\boldsymbol{x}$*), then*

$$\lim_{t\to\infty} \boldsymbol{g}^{t}_{168}(\boldsymbol{x}) = (\overset{j+1}{\mathbf{0}}, \mathbf{1}) \ .$$

(2-iv) For a configuration $\boldsymbol{x}$ *except for the above patterns, we have*

$$\lim_{t\to\infty} \boldsymbol{g}^{t}_{168}(\boldsymbol{x}) = \mathbf{0} \ .$$

$(\boldsymbol{g}_{168}, \mathcal{S}_{0(1),1(\leq k)})$ is a sub-dynamical system of $(\boldsymbol{g}_{168}, \mathcal{A})$ and a left-shift dynamical system. Following the way of the proof similar to that of F.Ohi (2007), we have the next theorem.

Theorem 3.2. *For every* $k \geq 2$*,* $(\mathcal{S}_{0(1),1(\leq k)}, \boldsymbol{g}_{168})$ *is Devaney chaos.*

Reminding that

$$\mathcal{S}_{0(1)} = \cup_{k=1}^{\infty} \mathcal{S}_{0(1),1(\leq k)} = \{ \ (0, \mathbf{1}_{k_j})_{j=-\infty}^{\infty} \mid k_j \geq 1, \ \sup_j k_j < \infty \ \}$$

is the set of configurations consisted of $(0, \mathbf{1}_{k_j})$, $(-\infty < j < \infty)$, where the length of the 1 blocks are bounded and every 0 state site is isolated, $\boldsymbol{g}_{168}$ is the left shift transformatio on $\mathcal{S}_{0(1)}$ and we have the folowing Theorem.

Theorem 3.3. $(\mathcal{S}_{0(1)}, \boldsymbol{g}_{168})$ *is Devanery chaos.*

Letting $\mathcal{S}$ denote the set of all configurations of which every 0 state site is isolated. $\mathcal{S}_{0(1)}$ is a proper subset of $\mathcal{S}$, $\mathcal{S}_{0(1)} \subset \mathcal{S}, \mathcal{S}_{0(1)} \neq \mathcal{S}$, since the length of the 1 blocks in a configuration of $\mathcal{S}$ are not necessarily bounded.

$(\mathcal{S}, \boldsymbol{g}_{168})$ is a left shift dynamical system and we have also the following theorem.

Theorem 3.4. $(\mathcal{S}, \boldsymbol{g}_{168})$ *is Devaney chaos.*

The above two theorems are almost clear, since for any configuration $\boldsymbol{x} \in \mathcal{S}$, every pattern of $(x_{-k}, \cdots, x_k)$ can be consisited of at most $01, 011, 0111$ and $0\underbrace{1 \cdots 1}_{2k+1}$, then the set of all periodic configurations is shown to be dense and a dense orbit is easily constructed.

The following theorem about periodic configurations of rule 168 is clear.

Theorem 3.5. *(1)* $(\boldsymbol{g}_{168}, \mathcal{S}_{0(1),1(1,2)})$ *has every periodic configuration except for the ones having prime period* 1, 4 *or* 6.

(2) $(\boldsymbol{g}_{168}, \mathcal{S}_{0(1),1(1,2,3)})$ *has every periodic configuration except for ones having the prime period* 1 *or* 4.

(3) $(\boldsymbol{g}_{168}, \mathcal{S}_{0(1),1(\leq k)})$ $(k \geq 4)$ *has every periodic configuration except for ones having prime period* 1.

(4) For every $(k \geq 4)\text{ÅC}$ $(\boldsymbol{g}_{168}, \mathcal{S}_{0(1),1(\leq k)} \cup \{\mathbf{0}\})$ *has every periodic configuration.*

Remark 3.1. (1) Since rule 224 is dual of rule 168, then the dual theorems of the above ones hold for rule 224.

(2) $(\boldsymbol{g}_{168}, \mathcal{S}_{0(1),1(1,2)})$ is equivalent to $(\boldsymbol{g}_{40}, \mathcal{S}_{0(1),1(1,2)})$, since both of them are left-shift dynamical systems on the same set $\mathcal{S}_{0(1),1(1,2)}$, then referring Theorem 3, we might say that rule 168 involves rule 40.

4 Right spreading rate of rule 168

Without loss of generality we may assume for $\boldsymbol{x} \in \mathcal{S}_{0(1)}$

$$\boldsymbol{x} = (\cdots, \overset{0}{\smile}{1}, 0, \mathbf{1}_{m_1}, 0, \mathbf{1}_{m_2}, 0, \mathbf{1}_{m_3}, 0, \cdots) \ .$$

Letting

$$n_0(\boldsymbol{x}) = (\cdots, \overset{\overset{0}{\smile}}{0}, 0, \mathbf{1}_{m_1}, 0, \mathbf{1}_{m_2}, 0, \mathbf{1}_{m_3}, 0, \cdots)\ ,$$

we examine the time development of the right-most different site of $\boldsymbol{x}$ and $n_0(\boldsymbol{x})$. From Lemma 3 (1) and Theorem 2 (1) we have,

$$\begin{aligned}
\boldsymbol{g}^{m_1}(\boldsymbol{x}) &= (\cdots, 1, 0, \overset{\overset{1}{\smile}}{\mathbf{1}_{m_1}}, \overset{\overset{2}{\smile}}{0}, \mathbf{1}_{m_2}, 0, \mathbf{1}_{m_3}, 0, \cdots)\ ,\\
\boldsymbol{g}^{m_1+1}(\boldsymbol{x}) &= (\cdots, 1, 0, \overset{\overset{0}{\smile}}{\mathbf{1}_{m_1}}, \overset{\overset{1}{\smile}}{0}, \overset{\overset{2}{\smile}}{1}, \mathbf{1}_{m_2-1}, 0, \mathbf{1}_{m_3}, 0, \cdots)\ ,\\
\boldsymbol{g}^{m_1}(n_0(\boldsymbol{x})) &= (\cdots, \overset{\overset{0}{\smile}}{0}, \overset{\overset{1}{\smile}}{0}, \overset{\overset{2}{\smile}}{0}, \mathbf{1}_{m_2}, 0, \mathbf{1}_{m_3}, 0, \cdots)\ ,\\
\boldsymbol{g}^{m_1+1}(n_0(\boldsymbol{x})) &= (\cdots, \overset{\overset{0}{\smile}}{0}, \overset{\overset{1}{\smile}}{0}, \overset{\overset{2}{\smile}}{0}, \mathbf{1}_{m_2-1}, 0, \mathbf{1}_{m_3}, 0, \cdots)\ ,
\end{aligned}$$

then the coordinate numbers of the right-most different sites are given in the following.

$$\begin{aligned}
1 \le t \le m_1,&\ \ DFR(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(n_0(\boldsymbol{x})) = 1,\\
m_1 + 1 \le t \le m_1 + m_2,&\ \ DFR(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(n_0(\boldsymbol{x})) = 2,\\
m_1 + m_2 + 1 \le t \le m_1 + m_2 + m_3,&\ \ DFR(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(n_0(\boldsymbol{x})) = 3.
\end{aligned}$$

Hence we have inductively

$$m_1 + \cdots + m_n + 1 \le t \le m_1 + \cdots + m_n + m_{n+1}\ ,$$
$$\frac{DFR(\boldsymbol{g}^t(\boldsymbol{x}), \boldsymbol{g}^t(n_0(\boldsymbol{x}))}{t} = \frac{n}{t},$$

and if the limiting value

$$\lim_{n\to\infty} \frac{n}{m_1 + \cdots + m_n}$$

exists, then the value is the right spreading rate of the configuration $\boldsymbol{x}$.

Letting $(0, \mathbf{1}_k, 0)(\boldsymbol{x}_{0,j})$ be the number of $(0, \mathbf{1}_k, 0)$ contained in $\boldsymbol{x}_{0,j}$, the right spreading rate is the limiting value

$$\lim_{j\to\infty} \frac{\sum_{n=1}^{\infty} (0, \mathbf{1}_n, 0)(\boldsymbol{x}_{0,j})}{\sum_{n=1}^{\infty} n \cdot (0, \mathbf{1}_n, 0)(\boldsymbol{x}_{0,j})}\ .$$

The infinite sums in the denominator and numerator are practically finite, since $\boldsymbol{x}_{0,j}$ consists of finite number of terms.

Let the configuration $\boldsymbol{x}$ be a member of $\mathcal{S}_{0(1),1(\le k)}$ $(k \ge 2)$.

Then the right spreading rate is determined by the limiting value of

$$\lim_{j\to\infty} \frac{\sum_{n=1}^{k} (0, \mathbf{1}_n, 0)(\boldsymbol{x}_{0,j})}{\sum_{n=1}^{k} n \cdot (0, \mathbf{1}_n, 0)(\boldsymbol{x}_{0,j})} = \lim_{j\to\infty} \frac{\sum_{n=1}^{k} \dfrac{(0, \mathbf{1}_n, 0)(\boldsymbol{x}_{0,j})}{j}}{\sum_{n=1}^{k} n \cdot \dfrac{(0, \mathbf{1}_n, 0)(\boldsymbol{x}_{0,j})}{j}},$$

which means that the relative frequency of $(0, \mathbf{1}_n, 0)$ in $\boldsymbol{x}_{0,\rightarrow}$ $1 \leq n \leq k$ determine the right spreading rate of $\boldsymbol{x}$.

For $\boldsymbol{x} \in \mathcal{S}_{0(1),1(\leq k)}$ it is sufficient to focus on $\boldsymbol{x}_{0,\rightarrow}$. Then we examine the dynamical system $(\mathcal{S}^{+}_{0(1),1(\leq k)}, \sigma_L)$ which is topologically conjugate to the interval dynamical system $([0,1], f)$, where the graph of the mapping $f : [0,1] \rightarrow [0,1]$ is given by the Fig. 1.

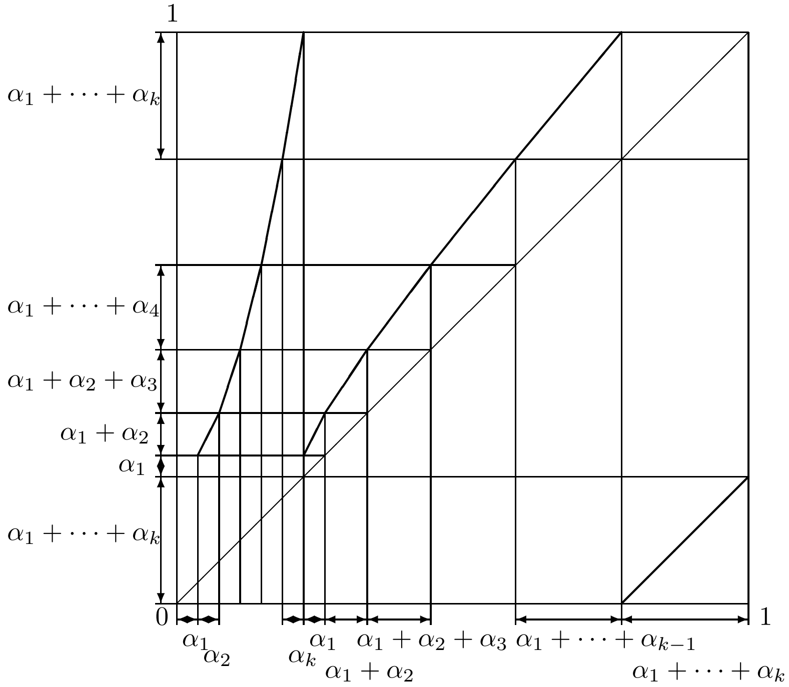

1
$\alpha_1 + \cdots + \alpha_k$
$\alpha_1 + \cdots + \alpha_4$
$\alpha_1 + \alpha_2 + \alpha_3$
$\alpha_1 + \alpha_2$
α_1
$\alpha_1 + \cdots + \alpha_k$
0
1
α_1
α_2
α_k
α_1
$\alpha_1 + \alpha_2$
$\alpha_1 + \alpha_2 + \alpha_3$
$\alpha_1 + \cdots + \alpha_{k-1}$
$\alpha_1 + \cdots + \alpha_k$

Fig. 1. An interval dynamical system $([0,1], f)$ conjugate to $(\mathcal{S}^{+}_{0(1),1(\leq k)}, \sigma_L)$

The homeomorphism $\Psi : [0,1] \rightarrow \mathcal{S}^{+}_{k}$ is defined as

$$x \in [0,1]\ , \quad (\Psi(x))_i = \begin{cases} 0, \ f^i(x) \in [\ 0\ ,\ \sum_{j=1}^{k} \alpha_j\ ), \\ 1, \ f^i(x) \in [\ \sum_{j=1}^{k} \alpha_j\ ,\ 1\ ]. \end{cases}$$

The Lebegue measure $\boldsymbol{l}$ on $([0,1], \mathcal{B}_{[0,1]})$ is easily shown to be the invariant measure of $([0,1], f)$ and also Ergodic by the Folklore Theorem (A. Boyarski, et al. [1]). The induced measure of $\boldsymbol{l}$ by Ψ is an Ergodic measure on $\mathcal{S}^{+}_{0(1),1(\leq k)}$. Relative frequencies by this Ergodic measure is the relative frequencies derived on $([0,1], f, \boldsymbol{l})$ as the following.

Diverting the symbol $(0, \mathbf{1}_n, 0)$, we interpret the symbol as a function of $n+2$ variable.

$$\begin{aligned}&(0, \mathbf{1}_n, 0)(a_0, a_1, \cdots, a_i, a_{n+1})\\ &\qquad = \begin{cases} 1, \, a_0 = 0, a_1 = \cdots = a_i = 1, a_{n+1} = 0, \\ 0, \text{ otherwise.} \end{cases}\end{aligned}$$

It is sufficient to derive the limiting value

$$\lim_{j \to \infty} \frac{\sum_{n=1}^{k} \frac{\sum_{i=0}^{j} (0, \mathbf{1}_n, 0)(f^i(x), f^{i+1}(x), \cdots, f^{i+n+1}(x))}{j}}{\sum_{n=1}^{k} n \cdot \frac{\sum_{i=0}^{j} (0, \mathbf{1}_n, 0)(f^i(x), f^{i+1}(x), \cdots, f^{i+n+1}(x))}{j}} \; .$$

By Ergodic theorem, we have

$$\begin{aligned}&\frac{\sum_{i=0}^{j} (0, \mathbf{1}_n, 0)(f^i(x), f^{i+1}(x), \cdots, f^{i+n+1}(x))}{j}\\ &\qquad\qquad \to \mathbf{E}[(0, \mathbf{1}_n, 0)(\cdot, f^1(\cdot), \cdots, f^{n+1}(\cdot))], \quad \boldsymbol{l} - a.s.\end{aligned}$$

Noticing

$$\mathbf{E}[(0, \mathbf{1}_n, 0)(\cdot, f^1(\cdot), \cdots, f^{n+1}(\cdot))] = \alpha_{k-n+1} \; ,$$

the right spreading rate we want to have is

$$\frac{\sum_{n=1}^{k} \alpha_n}{\sum_{n=1}^{k} (k - n + 1) \cdot \alpha_n} .$$

α_n $(n = 1, \cdots, k)$ should satisfy

$$\sum_{i=1}^{k} \sum_{j=1}^{i} \alpha_j + \sum_{j=1}^{k} \alpha_j = 1 \; , \quad \alpha_j > 0 \; , \; 1 \le j \le k \; .$$

The next equality and inequalities are easily verified.

$$\frac{1}{k} < \frac{\sum_{i=1}^{k} \alpha_i}{\sum_{i=1}^{k} (k - i + 1) \cdot \alpha_i} = \frac{\sum_{i=1}^{k} \alpha_i}{1 - \sum_{i=1}^{k} \alpha_i} \le 1 \; .$$

Thus we have the following theorem about the right spreading rate.

Theorem 4.1. *For rule 168, for arbitrarily given real number between 0 and 1, there exists a configuration of which right spreading rate is the given number.*

Remark 4.1. (1) The left spreading rate of every configuration $\boldsymbol{x} \in \mathcal{S}_{0(1)}$ is 1, which is easily proved.

(2) For rule 168, spreading rate and Lyapunov exponent are equivalent with each other.

5 Concluding remarks

In this paper we have shown that rule 168 of Wolfram class I defines a chaotic dynamical system and right spreading rates of configurations of specific type, which are also Lyapunov's exponents of the configurations, are the expectation of relative frequencies of some patterns observed in a topologically conjugate interval dynamical system $([0,1], f)$.

Rule 168 defines a chaotic dynamical system which is simply a left shift dynamical system and cannot be given in a computer simulation, since 0 state site in a configuration of $\mathcal{S}_{0(1)}$ is isolated and Bernoulli measure cannot give such a configuration as an initial one.

Reminding that for a randomly given initial configuration the time-space pattern generated by a member of Wolfram's class II is said to be simple as periodic, it might be interesting that rule 56 of the class II defines a chaotic dynamical system more complicated than left or right shift dynamical system. We are preparing a paper about full examination of rule 56.

References

[1] A. Boyarsky and P. Gora, Laws of Chaos, Invariant Measures and Dynamical Systems in One Dimension, Birkhauser (1997) p. 110–118.

[2] G. Cattaneo, E.Formenti and L.Margara, Topological definitions of deterministic chaos, In Cellular Automata, eds. M Delorme and J. Mazoyer, Kluwer Acadenc Publishers (1999), 213–259.

[3] A. Ilachinski, Cellular Automata — A Discrete Universe. World Scientific (2001).

[4] F. Ohi, Chaotic properties of the elementary cellular automaton rule 40 in Wolfram's class I, Complex Systems 17 (2007) 295–308

[5] M. A. Shereshevsky, Lyapunov exponents for one-dimensional cellular Automata, J. Nonlinear Sci. 2 (1992) 1–8.

[6] S. Wolfram, Statistical mechanics of cellular automata. Review of Modern Physics **55** (1983) 601–644.

[7] S. Wolfram, Universality and complexity in cellular automata Physica **10D** (1984) 1–35.

[8] S. Wolfram, A New Kind of Science. Wolfram Media, Inc. (2002).

Real-time reversible language recognition by cellular automata[⋆]

Martin Kutrib and Andreas Malcher

Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,malcher}@informatik.uni-giessen.de

Abstract. We investigate cellular automata and iterative arrays as acceptors for formal languages. In particular, we consider real-time devices which are reversible on the core of computation, i.e., from initial configuration to the configuration given by the time complexity t. This property is called t-time reversibility. We study whether for a given real-time device there exists a reverse real-time device with the same neighborhood. It is shown that real-time reversible iterative arrays can simulate restricted variants of stacks and queues. It turns out that real-time reversible iterative arrays are strictly weaker than real-time reversible cellular automata. On the other hand, they accept non-semilinear languages. We show that real-time reversibility itself is not even semidecidable, which extends the undecidability for cellular automata and contrasts the general case, where reversibility is decidable for one-dimensional devices. Moreover, we prove the undecidability of several other properties. Several closure properties are also derived.

1 Introduction

Reversibility in the context of computing devices means that deterministic computations are also backward deterministic. Roughly speaking, in a reversible device no information is lost and every configuration occurring in any computation has at most one predecessor. Many different formal models have been studied in connection with reversibility. For example, reversible Turing machines have been introduced in [3], where it is shown that any irreversible Turing machine can be simulated by a reversible one. With respect to the number of tapes and tape symbols the result is significantly improved in [22]. On the opposite end of the automata hierarchy, reversibility in very simple devices, namely deterministic finite automata, has been studied in [2] and [28].

Here we study linear arrays of identical copies of deterministic finite automata. The single nodes, except the node at the origin, are homogeneously connected to its both immediate neighbors. Moreover, they work synchronously

[⋆] Summary of the full papers [18, 19] whose extended abstracts have been presented at LATA 2007 and FCT 2007 at discrete time steps. The distinguished cell at the origin, the communication cell, is equipped with a one-way read-only input tape. Such devices are commonly called *iterative arrays* (IA). Closely related to iterative arrays are *cellular automata* (CA). Basically, the difference is that cellular automata receive their input in parallel. That is, in our setting the input is fed to the cells in terms of states during a pre-initial step. There is no extra input tape.

In connection with formal language recognition IAs have been introduced in [7]. In [9] a real-time acceptor for prime numbers has been constructed. A characterization of various types of IAs in terms of restricted Turing machines and several results, especially speed-up theorems, are given in [11]. Some recent results concern infinite hierarchies beyond linear time [13] and between real time and linear time [6], hierarchies depending on the amount of nondeterminism [5], and descriptional complexity issues [20]. It is well known that conventional *real-time* cellular automata are strictly more powerful than *real-time* iterative arrays [30]. Our particular interest lies in reversible devices as acceptors for formal languages. An early result on general reversible CAs is the possibility to make any CA, possibly irreversible, reversible by increasing the dimension. In detail, in [31] it is shown that any k-dimensional CA can be embedded into a $(k+1)$-dimensional reversible CA. Again, this result has significantly been improved by showing how to make irreversible one-dimensional CAs reversible without increasing the dimension [25]. A solution is presented which preserves the neighborhood but increases time ($O(n^2)$ time for input length n). Furthermore, it is known that even reversible one-dimensional one-way CAs are computationally universal [21, 23]. Once a reversible computing device is under consideration, the natural question arises whether reversibility is decidable. For example, reversibility of a given deterministic finite automaton or of a given regular language is decidable [28]. For cellular automata, injectivity of the global transition function is equivalent to the reversibility of the automaton. It is shown in [1] that global reversibility is decidable for one-dimensional CAs, whereas the problem is undecidable for higher dimensions [14]. Additional information on some aspects of CAs may be found in [15, 17]. All these results concern cellular automata with unbounded configurations. Moreover, in order to obtain a reversible device the neighborhood as well as the time complexity may be increased. In [8] it is shown that the neighborhood of a reverse CA is at most $n-1$ when the given reversible CA has n states. Additionally, this upper bound is shown to be tight. In connection with pattern recognition reversible two-dimensional partitioned cellular automata have been investigated in [24, 26]. A recent survey of reversibility in cellular automata is [27].

Here, in contrast to the traditional notion of reversibility, we consider cellular automata working on finite configurations with fixed boundary conditions. Clearly, these devices cannot be reversible in the classical sense since the number of different configurations is bounded and, thus, the system will run into loops. Therefore, we consider cellular automata and iterative arrays that are reversible on the core of computation, i.e., from initial configuration to the configuration given by the time complexity. Our main interest is in fast computations, i.e., real-time computations. Consequently, we call such devices real-time reversible. In particular, we want to know whether for a given nearest neighbor real-time device there exists a reverse real-time device with the same neighborhood. This point of view is rather different from the traditional notion of reversibility since only configurations are considered that are reachable from initial configurations. At first glance, such a setting should simplify matters. But quite the contrary, we prove that real-time reversibility is not even semidecidable for iterative arrays, which extends the undecidability for cellular automata and contrasts the general case, where reversibility is decidable for one-dimensional devices. Moreover, it is shown that emptiness is undecidable. Thus, also the questions of finiteness, infiniteness, inclusion, and equivalence are undecidable. The same holds true for regularity and context-freedom. We give evidence that real-time reversible iterative arrays can simulate restricted variants of stacks and queues. It turns out that real-time reversible iterative arrays are strictly weaker than real-time reversible cellular automata. On the other hand, a non-semilinear language is accepted. Finally, we present some results concerning closure properties under several operations.

2 Real-time reversible cellular automata and iterative arrays

We denote the set of non-negative integers by $\mathbb{N}$. The empty word is denoted by λ, and the reversal of a word w by w^R. For the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions. In order to avoid technical overloading in writing, two languages L and L' are considered to be equal, if they differ at most by the empty word, i.e., $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$.

We consider devices where each cell is connected to its both nearest neighbors, and identify the cells by positive integers.

Definition 2.1. *A* cellular automaton *(CA) is a system $\langle S, \delta, \#, A, F\rangle$, where S is the finite, nonempty set of* cell states*, $\# \notin S$ is the* boundary state*, $A \subseteq S$ is the nonempty set of* input symbols*, $F \subseteq S$ is the set of* accepting states*, and $\delta : (S \cup \{\#\}) \times S \times (S \cup \{\#\}) \to S$ is the* local transition function.

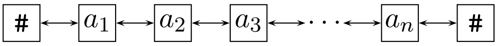


Fig. 1. A two-way cellular automaton.

A *configuration* of a cellular automaton $\mathcal{M} = \langle S, \delta, \#, A, F\rangle$ at time $t \geq 0$ is a description of its global state, which is actually a mapping $c_t : [1, \ldots, n] \to S$, for $n \geq 1$. The configuration at time 0 is defined by the initial sequence of states. For a given input $w = a_1 \cdots a_n \in A^+$ we define $c_{0,w}$ as the configuration at time 0 by $c_{0,w}(i) = a_i$, for $1 \leq i \leq n$. During its course of computation a CA steps through a sequence of configurations, whereby successor configurations are computed according to the global transition function Δ. Let c be some configuration, defined by $s_1, \dots, s_n \in S$, then the successor configuration c', defined by $s'_1, \dots, s'_n \in S$, is as follows:

$$c' = \Delta(c) \iff s'_1 = \delta(\#, s_1, s_2), s'_2 = \delta(s_1, s_2, s_3), \dots, s'_n = \delta(s_{n-1}, s_n, \#).$$

Thus, Δ is induced by δ.

An iterative array is a semi-infinite array of cells, where the leftmost cell at the origin, the communication cell, is connected to its right neighbor and, additionally, equipped with a one-way read-only input tape. At the outset of a computation the input is written with an infinite number of end-of-input symbols to the right on the input tape, and all cells are in the so-called quiescent state. The state transition of the communication cell additionally depends on the current input symbol. The head of the one-way input tape is moved at any step to the right.

Definition 2.2. *An* iterative array *(IA) is a system* $\langle S, A, F, s_0, \lhd, \delta, \delta_0 \rangle$*, where* S *is the finite, nonempty set of* cell states*,* A *is the finite, nonempty set of* input symbols*,* $F \subseteq S$ *is the set of* accepting states*,* $s_0 \in S$ *is the* quiescent state*,* $\lhd \notin A$ *is the* end-of-input symbol*,* $\delta : S^3 \to S$ *is the* local transition function for non-communication cells *satisfying* $\delta(s_0, s_0, s_0) = s_0$*,* $\delta_0 : (A \cup \{\lhd\}) \times S^2 \to S$ *is the* local transition function for the communication cell*.*

Let $\mathcal{M}$ be an IA. A configuration of $\mathcal{M}$ at some time $t \geq 0$ is a pair (w_t, c_t), where $w_t \in A^*$ is the remaining input sequence and $c_t : \mathbb{N} \to S$ is a mapping that maps the single cells to their current states. The configuration (w_0, c_0) at time 0 is defined by the input word w_0 and the mapping $c_0(i) = s_0$, $i \geq 0$, while subsequent configurations are chosen according to the global transition function Δ: Let (w_t, c_t), $t \geq 0$, be a configuration, then its successor configuration (w_{t+1}, c_{t+1}) is as follows:

$$(w_{t+1}, c_{t+1}) = \Delta\big((w_t, c_t)\big) \iff \begin{cases} c_{t+1}(i) = \delta\big(c_t(i-1), c_t(i), c_t(i+1)\big), i \geq 1, \\ c_{t+1}(0) = \delta_0\big(a, c_t(0), c_t(1)\big) \end{cases},$$

where $a = \lhd$, $w_{t+1} = \lambda$ if $w_t = \lambda$, and $a = a_1$, $w_{t+1} = a_2 \cdots a_n$ if $w_t = a_1 \cdots a_n$. Thus, the global transition function Δ is induced by δ and δ_0.

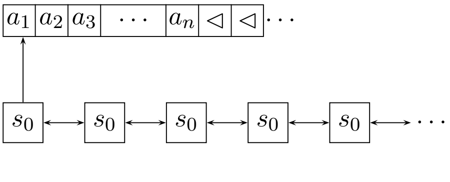


Fig. 2. Initial configuration of an iterative array.

An input w is accepted by a CA (IA) $\mathcal{M}$ if at some time i during its course of computation the leftmost cell enters an accepting state. $L(\mathcal{M}) = \{w \in A^* \mid w \text{ is accepted by } \mathcal{M}\}$ is the *language accepted* by $\mathcal{M}$. Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$ ($t(n) \geq n+1$ for IAs) be a mapping. If all $w \in L(\mathcal{M})$ are accepted with at most $t(|w|)$ time steps, then $L(\mathcal{M})$ is said to be of time complexity t.

Now we turn to cellular automata and iterative arrays that are reversible on the core of computation, i.e., from initial configuration to the configuration given by the time complexity. We call them t-time reversible, if the time complexity t is obeyed. Reversibility is meant with respect to the possibility of stepping the computation back and forth. Due to the domain S^3 and the range S, obviously, the local transition function cannot be injective in general. But for reverse computation steps we may utilize the information which is available for the cells, that is, the states of their neighbors. So, we have to provide a reverse local transition function.

For some mapping $t : \mathbb{N} \to \mathbb{N}$ let $\mathcal{M} = \langle S, \delta, \#, A, F \rangle$ be a t-time cellular automaton. Then $\mathcal{M}$ is defined to be *t-reversible* (REV-CA), if there exists a reverse local transition function $\delta_R : S^3 \to S$ such that $\Delta_R(\Delta(c_i)) = c_i$, for all configurations c_i of $\mathcal{M}$, $0 \leq i \leq t(n) - 1$. The global transition functions Δ and Δ_R are induced by δ and δ_R.

Similarly, let $\mathcal{M} = \langle S, A, F, s_0, \triangleleft, \delta, \delta_0 \rangle$ be a t-time iterative array. Then $\mathcal{M}$ is defined to be *t-reversible* (REV-IA), if there exist reverse local transition functions δ_R and δ_0^R such that $\Delta_R(\Delta(c_i)) = c_i$, for all configurations c_i of $\mathcal{M}$, $0 \leq i \leq t(n) - 1$. The global transition functions Δ and Δ_R are induced by δ, δ_0 and δ_R, δ_0^R. The head of the input tape is always moved to the *left* when the reverse transition function is applied. For distinctness, we denote the devices with reverse transition functions by $\mathcal{M}_R$.

The family of languages accepted by some REV-CA (REV-IA) with time complexity t is denoted by $\mathscr{L}_t(\text{REV-CA})$ ($\mathscr{L}_t(\text{REV-IA})$). If t equals the *identity function* $\text{id}(n) = n$ (the function $n+1$), acceptance is said to be in *real time* and we write $\mathscr{L}_{rt}(\text{REV-CA})$ ($\mathscr{L}_{rt}(\text{REV-IA})$).

In order to introduce some of the particularities in connection with reversible language recognition we continue with an example. The goal is to define a real-time REV-IA that accepts the non-context-free language $\{a^{n^2} b^{2n-1} \mid n \geq 1\}$, which is not even semilinear. We start with a conventional iterative array. Basically, the idea is to recognize time steps which are square numbers. To this end, assume k cells of the array are marked at time k^2. Then a signal can be emitted by the communication cell. The signal moves through the marked area, extends it by one cell, and moves back again. So, the signal arrives at the communication cell at time $(k+1)^2$. Finally, the number of bs is checked by sending another signal through the marked area and back. Figure 3 (left) shows an accepting computation. But the transition functions have to be extended in order to reject words not belonging to the language. To this end, we consider possible errors and observe that all errors can be detected by the communication cell. We identify the following errors: (1) the first input symbol is a b, (2) an a follows b, (3) the number of as is not a square number, (4) the number of bs is insufficient, or (5) there are too many bs. Accordingly, we provide rules to cope with the situations. An example of a rejecting computation is given in Fig. 3 (middle). Moreover, in our current construction the whole computation may get frozen before time step $n+1$, for inputs not belonging to the language. Clearly, this implies non-reversibility. One reason is that for conventional computations we do not care about rejecting computations, except for keeping them rejecting. Nor do we care about the part of the computation that cannot influence the overall result, that is, the computation of cells $i \geq 1$ after time step $n+1-i$, i.e., the area below the diagonal starting from the lower left corner of the space-time diagram.

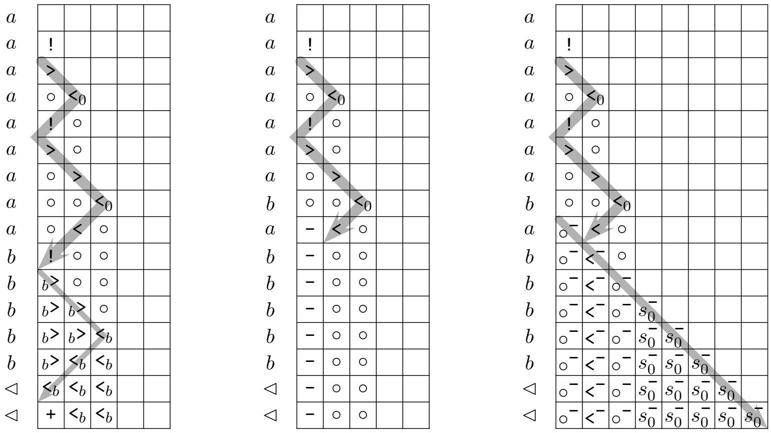


Fig. 3. Space-time diagrams of a real-time IA accepting $\{a^{n^2}b^{2n-1} \mid n \geq 1\}$ (left), not being reversible (middle), rejecting reversibly (right). Cells in quiescent state are left blank.

For reversible computations we do have to care about rejecting computations as well as for computations in the mentioned area. The idea of our construction is to send a signal from left to right which freezes the computation, whereby each cell passed through has to remember its current state. Clearly, this idea does not work in general. Sometimes much more complicated computations are necessary in order to obtain reversible computations. Next, we present the complete transition functions of a REV-IA accepting the language $\{a^{n^2}b^{2n-1} \mid n \geq 1\}$. For convenience, $\delta(p,q,r) = s$ is written as $pqr \to s$, and the same holds for δ_0. By x, z we denote arbitrary states.

δ_0	δ	δ_0	δ
$a\ s_0\ s_0 \rightarrow\ !$	$>\ s_0\ s_0 \rightarrow\ <_0$	$b\ s_0\ s_0 \rightarrow s_0{}^-$	$>^-\ s_0\ s_0 \rightarrow s_0^-$
$a\ !\ z \rightarrow\ >$	$>\ \circ\ y \rightarrow\ >$	$a\ {}_b{>}\ z \rightarrow {}_b{>}^-$	$>^-\ \circ\ y \rightarrow \circ^-$
$a\ >\ z \rightarrow \circ$	$x\ \circ\ <_0 \rightarrow\ <$	$a\ <_b\ z \rightarrow\ <_b^-$	$x^-\ \circ\ <_0 \rightarrow \circ^-$
$a\ \circ\ < \rightarrow\ !$	$x\ \circ\ < \rightarrow\ <$	$b\ \circ\ z \rightarrow \circ^-$	$x^-\ \circ\ < \rightarrow \circ^-$
$a\ \circ\ <_0 \rightarrow\ !$	$x\ >\ z \rightarrow \circ$	$b\ >\ z \rightarrow\ >^-$	$x^-\ >\ z \rightarrow\ >^-$
$b\ !\ s_0 \rightarrow\ <_b$	$x\ <_0\ s_0 \rightarrow \circ$	$b\ <_b\ z \rightarrow\ <_b^-$	$x^-\ <_0\ s_0 \rightarrow\ <_0^-$
$b\ !\ \circ \rightarrow {}_b{>}$	$x\ <\ z \rightarrow \circ$	$\lhd\ s_0\ z \rightarrow s_0^-$	$x^-\ <\ z \rightarrow\ <^-$
$b\ {}_b{>}\ <_b \rightarrow\ <_b$	${}_b{>}\ \circ\ \circ \rightarrow {}_b{>}$	$\lhd\ !\ z \rightarrow\ !^-$	${}_b{>}^-\ \circ\ \circ \rightarrow \circ^-$
$\lhd\ <_b\ z \rightarrow +$	${}_b{>}\ \circ\ s_0 \rightarrow\ <_b$	$\lhd\ >\ z \rightarrow\ >^-$	${}_b{>}^-\ \circ\ s_0 \rightarrow \circ^-$
	$x\ {}_b{>}\ <_b \rightarrow\ <_b$	$\lhd\ \circ\ z \rightarrow \circ^-$	$x^-\ {}_b{>}\ <_b \rightarrow {}_b{>}^-$
		$\lhd\ {}_b{>}\ z \rightarrow {}_b{>}^-$	$x^-\ s_0\ s_0 \rightarrow s_0^-$

The two blocks of transition rules at the left are for accepting computations. The third block provides rules for detecting that the input is of wrong format. The rules of the fourth block are for the freezing error signal. An example for a reversible rejecting computation is given in Fig. 3 (right). It is not hard to obtain the reverse local transition functions δ_0^R and δ_R.

Concerning the computational capacity of IA it is well known and one of the fundamental results that conventional real-time cellular automata are strictly more powerful than real-time iterative arrays [30]. The next theorem establishes a reversible relationship equal to the relationship in the conventional case (cf. Fig. 4).

Theorem 2.1. *The family $\mathscr{L}_{rt}$(REV-IA) is properly included in the family of languages accepted by real-time reversible cellular automata.*

3 Reversible simulation of data structures

We next want to explore the computational capacity of real-time REV-IAs in more detail. To this end, we first consider the data structures *stack* and *queue*, and show that REV-IAs can simulate special variants thereof. We start with the stack. In detail, we consider real-time deterministic pushdown automata accepting linear context-free languages. Moreover, the stack behavior is restricted in such a way that in every step exactly one symbol is pushed on or popped from the stack. For convenience, we denote the family of languages accepted by such automata by DLR.

Theorem 3.1. *Every language from DLR belongs to the family $\mathscr{L}_{rt}$(REV-IA).*

Now we can utilize the simulation (cf. Fig. 5) in order to derive particular languages belonging to the family $\mathscr{L}_{rt}$(REV-IA).

Example 3.1. Every regular language as well as the languages $\{a^n b^n \mid n \geq 1\}$ and $\{wccw^R \mid w \in \{a,b\}^+\}$ are in DLR and, thus, belong to $\mathscr{L}_{rt}$(REV-IA).

Similar to the restricted stack behavior, we can show that real-time REV-IAs can simulate queues under certain conditions.

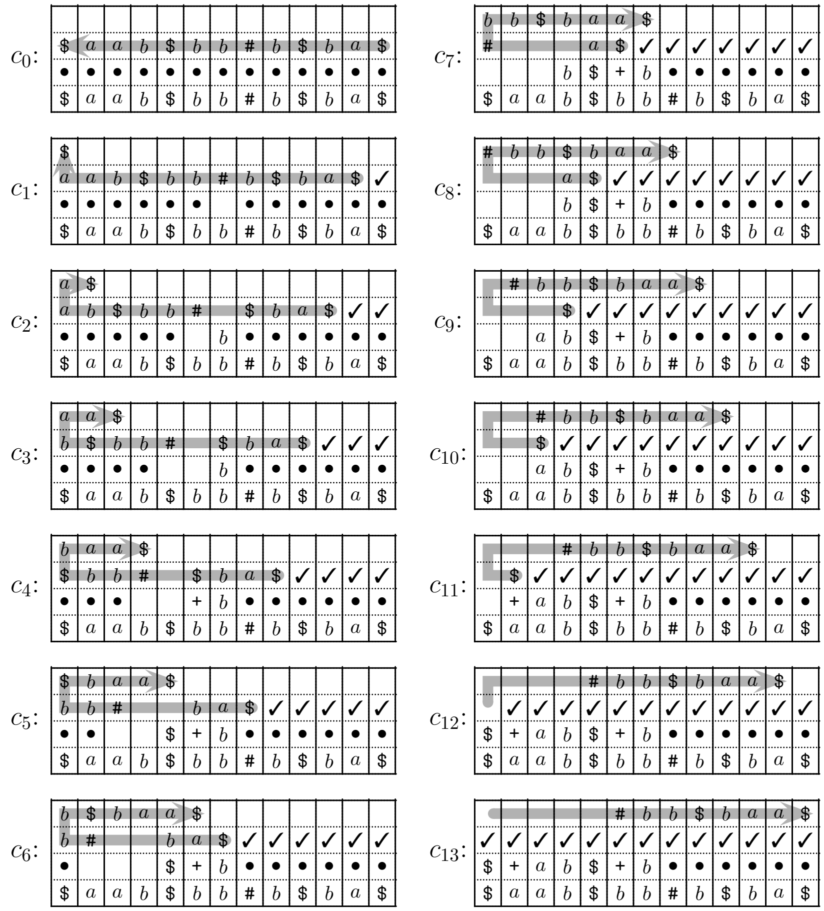


Fig. 4. Example computation of a real-time reversible cellular automaton accepting $\{\$x_k\$\cdots\$x_1\#y_1\$\cdots\$y_k\$ \mid 1 \leq k, x_i^R = y_i z_i, x_i, y_i, z_i \in \{a,b\}^*, 1 \leq i \leq k\}$ which cannot be accepted by any conventional real-time iterative array.

Lemma 3.1. *Let $\mathcal{Q}$ be an empty queue which is filled by a number of* `in` *operations, and then emptied by a sequence of* `out` *operations. Moreover, in every time step exactly one* `in` *or* `out` *operation is performed. Then $\mathcal{Q}$ can be simulated by a real-time REV-IA.*

Example 3.2. Consider the language $L = \{wcwc \mid w \in \{a,b\}^+\}$. First, the input prefix wc is inserted into a queue. Then, the content of the queue is removed step by step, whereby it is matched against the remaining input wc. Due to Lemma 3.1, language L belongs to the family $\mathscr{L}_{rt}$(REV-IA).

The restrictions on the data structures seem to be very natural, since the deterministic, linear context-free language $L = \{\$x_k\$\cdots\$x_1\#y_1\$\cdots\ \$y_k\$ \mid x_i^R =$

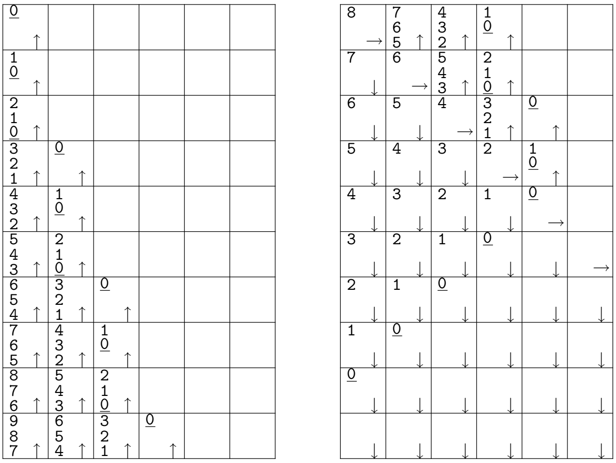


Fig. 5. Pushing (left) and popping (right) of ten pushdown symbols in real time. The left half of each cell contains the three registers for simulating the stack. The first two registers of the right half are used to store the current state of the communication cell and the last popped stack symbol, respectively. The last register indicates whether the stack is increasing (↑), decreasing (↓), or a switch takes place (→). The first entry of the stack is marked by underlining.

$y_i z_i, x_i, y_i, z_i \in \{a, b\}^*\}$ used as witness in the proof of Theorem 2.1 is not accepted by any conventional real-time iterative array.

4 Closure properties

The technique to send a signal that freezes the computation in order to maintain reversibility in certain situations, yields the closure of the family $\mathscr{L}_{rt}$(REV-IA) under Boolean operations. A family of languages is said to be *effectively* closed under some operation, if the result of the operation can be constructed from the given language(s).

Lemma 4.1. *The family $\mathscr{L}_{rt}$(REV-IA) is effectively closed under the Boolean operations complementation, union, and intersection.*

Next, we want to show closure under inverse homomorphism. The closure of languages accepted by conventional real-time iterative arrays has similarly been obtained in [29]. We start with some preliminaries.

Let A and B be two alphabets. The shuffle of two words $x \in A^*$ and $y \in B^*$ is $x \sqcup\!\sqcup\, y = \{x_1 y_1 \dots x_k y_k \mid x = x_1 \dots x_k, y = y_1 \dots y_k, 1 \leq i \leq k, k \geq 1\}$. The shuffle of two languages $L \subseteq A^*$ and $L' \subseteq B^*$ is defined as $L \sqcup\!\sqcup\, L' = \{x \sqcup\!\sqcup\, y \mid x \in L \text{ and } y \in L'\}$.

Lemma 4.2. *Let A and B be two disjoint alphabets. If $L \subseteq A^*$ is accepted by a real-time REV-IA, then $L \sqcup\!\sqcup\, B^*$ is accepted by a real-time REV-IA as well.*

Lemma 4.3. *The family $\mathscr{L}_{rt}(REV\text{-}IA)$ is effectively closed under inverse homomorphism.*

Lemma 4.4. *The family $\mathscr{L}_{rt}(REV\text{-}IA)$ is effectively closed under marked concatenation and right concatenation with regular languages.*

We derive further closure properties using the above-mentioned language $L = \{\$x_k\$\cdots\$x_1\#y_1\$\cdots\$y_k\$ \mid 1 \leq k, x_i^R = y_i z_i, x_i, y_i, z_i \in \{a,b\}^*, 1 \leq i \leq k\}$.

Lemma 4.5. *The family $\mathscr{L}_{rt}(REV\text{-}IA)$ is not closed under reversal, left concatenation with regular languages, and λ-free homomorphism.*

Next we summarize the results for the family $\mathscr{L}_{rt}$(REV-CA).

Theorem 4.1. *The family $\mathscr{L}_{rt}(REV\text{-}CA)$ is effectively closed under complementation, union, and intersection.*

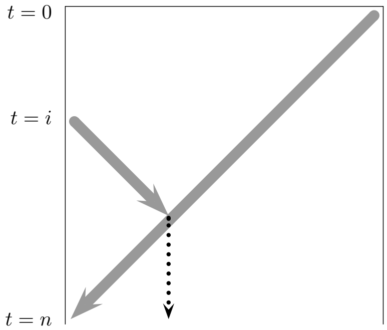


Fig. 6. Signals for the construction of a complementary real-time REV-CA. The leftmost cell enters an accepting state at time $t = i$ for the first time. The cell along the dotted line is marked.

5 Decidability questions

To show undecidability results for REV-CAs we use reductions from Post's correspondence problem (PCP) which is known to be undecidable. Let A be an alphabet and an instance of the PCP be given by two lists $\alpha = u_1, u_2, \ldots, u_k$ and $\beta = v_1, v_2, \ldots, v_k$ of words from A^+. Furthermore, let $A' = \{a_1, a_2, \ldots, a_k\}$ be an alphabet with k symbols and $A \cap A' = \emptyset$. Consider two languages L_α and L_β:

$$L_\alpha = \{u_{i_1}u_{i_2}\ldots u_{i_m}a_{i_m}a_{i_{m-1}}\cdots a_{i_1} \mid m \geq 1, 1 \leq i_j \leq k, 1 \leq j \leq m\}$$
$$L_\beta = \{v_{i_1}v_{i_2}\ldots v_{i_m}a_{i_m}a_{i_{m-1}}\cdots a_{i_1} \mid m \geq 1, 1 \leq i_j \leq k, 1 \leq j \leq m\}$$

Lemma 5.1. *The languages L_α and L_β belong to the family $\mathscr{L}_{rt}(REV\text{-}CA)$.*

We can utilize the languages of Lemma 5.1 to prove the first undecidable property of real-time REV-CAs.

Theorem 5.1. *Emptiness is undecidable for real-time REV-CAs.*

Lemma 5.2. *Let $\mathcal{M}$ be a real-time REV-CA and a, b, c be new alphabet symbols. Then the following languages belong to the family $\mathscr{L}_{rt}(REV\text{-}CA)$.*

1. $L_{\mathcal{M},1} = \{wa^{5|w|} \mid w \in L(\mathcal{M})\}$
2. $L_{\mathcal{M},2} = \{wa^{5|w|}(bc^{6|w|-1})^n \mid w \in L(\mathcal{M}), n \geq 0\}$
3. $L_{\mathcal{M},3} = \{wa^{|w|}b^{4|w|} \mid w \in L(\mathcal{M})\}$
4. $L_{\mathcal{M},4} = \{wa^{|w|}b^{|w|}c^{3|w|} \mid w \in L(\mathcal{M})\}$

Theorem 5.2. *Finiteness, infiniteness, universality, inclusion, equivalence, regularity, and context-freedom is undecidable for real-time REV-CAs.*

Theorem 5.3. *Let $\mathcal{M}$ be a real-time CA. It is undecidable whether or not $\mathcal{M}$ is real-time reversible.*

Now we turn to explore undecidable properties for real-time REV-IAs. To this end, we consider *valid computations of Turing machines* [10]. Roughly speaking, these are histories of accepting Turing machine computations. It suffices to consider deterministic Turing machines with a single tape and a single read-write head. Without loss of generality and for technical reasons, one can assume that any accepting computation has at least three and, in general, an odd number of steps. Therefore, it is represented by an even number of configurations. Moreover, it is assumed that the Turing machine cannot print blanks, and that a configuration is halting if and only if it is accepting. The language accepted by some machine $\mathcal{M}$ is denoted by $L(\mathcal{M})$.

Let S be the state set of some Turing machine M, where s_0 is the initial state, $T \cap S = \emptyset$ is the tape alphabet containing the blank symbol, $A \subset T$ is the input alphabet, and $F \subseteq S$ is the set of accepting states. Then a configuration of M can be written as a word of the form T^*ST^* such that $t_1 \cdots t_i s t_{i+1} \cdots t_n$ is used to express that M is in state s, scanning tape symbol t_{i+1}, and t_1 to t_n is the support of the tape inscription. The set of valid computations VALC($\mathcal{M}$) is now defined to be the set of words of the form $w_1\texttt{\#\#\#\#}w_2\texttt{\#\#\#\#}\cdots\texttt{\#\#\#\#}w_{2m}\texttt{\#\#\#\#}$, where $m \geq 2$, $\texttt{\#} \notin T \cup S$, $w_i \in T^*ST^*$ are configurations of M, w_1 is an initial configuration of the form s_0A^*, w_{2m} is an accepting configuration of the form T^*FT^*, and w_{i+1} is the successor configuration of w_i, with $0 \leq i \leq 2m-1$. The set of *invalid computations* INVALC(M) is the complement of VALC(M) with respect to the alphabet $\{\texttt{\#}\} \cup T \cup S$.

The following lemma is the key tool to prove undecidability properties for real-time REV-IAs.

Lemma 5.3. *Let $\mathcal{M}$ be a Turing machine. Then the set* VALC[$\mathcal{M}$] *can be represented as the intersection of two languages from $\mathscr{L}_{rt}(REV\text{-}IA)$.*

By Lemma 4.1 the family $\mathscr{L}_{rt}$(REV-IA) is closed under intersection. So, we obtain the following corollary.

Corollary 5.1. *Let $\mathcal{M}$ be a Turing machine. Then the set* VALC[$\mathcal{M}$] *belongs to the family $\mathscr{L}_{rt}$(REV-IA).*

Theorem 5.4. *Emptiness, finiteness, infiniteness, universality, inclusion, equivalence, regularity, and context-freedom are not semidecidable for real-time reversible IAs.*

Theorem 5.5. *Let $\mathcal{M}$ be a real-time IA. It is not semidecidable whether or not $\mathcal{M}$ is real-time reversible.*

References

[1] Amoroso, S., Patt, Y.N.: Decision procedures for surjectivity and injectivity of parallel maps for tesselation structures. J. Comput. System Sci. **6** (1972) 448–464

[2] Angluin, D.: Inference of reversible languages. J. ACM **29** (1982) 741–765

[3] Bennet, C.H.: Logical reversibility of computation. IBM Journal of Research and Development **17** (1973) 525–532

[4] Buchholz, Th., Kutrib, M.: Some relations between massively parallel arrays. Parallel Comput. **23** (1997) 1643–1662

[5] Buchholz, Th., Klein, A., Kutrib, M.: Iterative arrays with limited nondeterministic communication cell. Words, Languages and Combinatorics III, World Scientific Publishing (2003) 73–87

[6] Buchholz, Th., Klein, A., Kutrib, M.: Iterative arrays with small time bounds. Mathematical Foundations of Computer Science (MFCS 1998). LNCS 1893, Springer (2000) 243–252

[7] Cole, S.N.: Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE Trans. Comput. **C-18** (1969) 349–365

[8] Czeizler, E., Kari, J.: A tight linear bound on the neighborhood of inverse cellular automata. International Colloquium on Automata, Languages and Programming (ICALP 2005). LNCS 3580, Springer (2005) 410–420

[9] Fischer, P.C.: Generation of primes by a one-dimensional real-time iterative array. J. ACM **12** (1965) 388–394

[10] Hartmanis, J.: Context-free languages and Turing machine computations. Proc. Symposia in Applied Mathematics **19** (1967) 42–51

[11] Ibarra, O.H., Palis, M.A.: Some results concerning linear iterative (systolic) arrays. J. Parallel Distributed Comput. **2** (1985) 182–218

[12] Imai, K., Morita, K.: Firing squad synchronization problem in reversible cellular automata. Theoret. Comput. Sci. **165** (1996) 475–482

[13] Iwamoto, C., Hatsuyama, T., Morita, K., Imai, K.: Constructible functions in cellular automata and their applications to hierarchy results. Theoret. Comput. Sci. **270** (2002) 797–809

[14] Kari, J.: Reversibility and surjectivity problems of cellular automata. J. Comput. System Sci. **48** (1994) 149–182

[15] Kari, J.: Theory of cellular automata: a survey. Theoret. Comput. Sci. **334** (2005) 3–33

[16] Kutrib, M.: Automata arrays and context-free languages. Where Mathematics, Computer Science and Biology Meet. Kluwer Academic Publishers (2001) 139–148

[17] Kutrib, M.: Cellular Automata – A Computational Point of View. New Developments in Formal Languages and Applications. Springer (2008) chapter 6, to appear

[18] Kutrib, M., Malcher, A.: Fast reversible language recognition using cellular automata. Inform. Comput., to appear

[19] Kutrib, M., Malcher, A.: Real-time reversible iterative arrays. Theoret. Comput. Sci., submitted

[20] Malcher, A.: On the descriptional complexity of iterative arrays. IEICE Trans. Inf. Syst. **E87-D** (2004) 721–725

[21] Morita, K., Harao, M.: Computation universality of one dimensional reversible injective cellular automata. Trans. IEICE **E72** (1989) 758–762

[22] Morita, K., Shirasaki, A., Gono, Y.: A 1-tape 2-symbol reversible Turing machine. Trans. of the IEICE **E72** (1989) 223–228

[23] Morita, K.: Computation-universality of one-dimensional one-way reversible cellular automata. Inform. Process. Lett. **42** (1992) 325–329

[24] Morita, K., Ueno, S.: Parallel generation and parsing of array languages using reversible cellular automata. Int. J. Pattern Recog. and Artificial Intelligence **8** (1994) 543–561

[25] Morita, K.: Reversible simulation of one-dimensional irreversible cellular automata. Theoret. Comput. Sci. **148** (1995) 157–163

[26] Morita, K., Ueno, S., Imai, K.: Characterizing the ability of parallel array generators on reversible partitioned cellular automata. Int. J. Pattern Recog. and Artificial Intelligence **13** (1999) 523–538

[27] Morita, K.: Reversible computing and cellular automata – A survey. Theoret. Comput. Sci. **395** (2008) 101–131

[28] Pin, J.E.: On reversible automata. Theoretical Informatics (LATIN 1992). LNCS 583, Springer (1992) 401–416

[29] Seidel, S. R.: Language recognition and the synchronization of cellular automata. Technical Report 79-02, University of Iowa (1979)

[30] Smith III, A.R.: Real-time language recognition by one-dimensional cellular automata. J. Comput. System Sci. **6** (1972) 233–253

[31] Toffoli, T.: Computation and construction universality of reversible cellular automata. J. Comput. System Sci. **15** (1977) 213–231

Recent results on iterative arrays with small space bounds*

Andreas Malcher**[1], Carlo Mereghetti[2], and Beatrice Palano[2]

[1] Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
malcher@informatik.uni-giessen.de

[2] Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano
via Comelico 39/41, 20135 Milano, Italy
{mereghetti,palano}@dsi.unimi.it

Abstract. Iterative arrays (IAs) are a parallel computational model with a sequential processing of the input. They are one-dimensional arrays of interacting identical deterministic finite automata. In this paper, realtime-IAs with *sublinear* space bounds are used to accept formal languages. The existence of an infinite proper hierarchy of space complexity classes between logarithmic and linear space bounds is proved. Some decidability questions on logarithmically space bounded realtime-IAs are investigated. Furthermore, an optimal space lower bound for non-regular language recognition on realtime-IAs is shown.

1 Introduction

Iterative arrays (IAs, for short) are computational devices consisting of an array of identical deterministic finite automata — called cells — which themselves are homogeneously interconnected with their both neighbors. An IA reads the input sequentially via a distinguished *communication cell*. The state of each cell is changed at discrete time steps by applying its transition function synchronously. Cole [2] was the first who studied formal language aspects of IAs. A survey on such aspects may be found in [3]. Some recent results concern communication-restricted IAs [4, 12] and reversible IAs [5].

The *space* used by IAs considers, as a function of the input length, the number of cells activated along computations. In the general model, as many cells as the input is long may be used. Here, we consider realtime-IAs which are allowed to use only a *sublinear* amount of space. This paper summarizes some recent results which have been submitted to [8]. A previous version has been published as technical report [7].

As a main result, we exhibit an infinite proper hierarchy of classes of languages accepted between logarithmic and linear space bounds by realtime-IAs. For sublogarithmic space bounds, we obtain that only regular languages can be accepted. Finally, some decidability questions on logarithmically space bounded realtime-IAs are studied.

[*] Summary of results submitted to [8].

[**] Corresponding author.

2 Definitions

An iterative array consists of a linear array of identical deterministic finite state automata called *cells.* At the beginning, all cells are in a designated quiescent state q_0. Each cell is connected with its left and right neighbor, except clearly the leftmost cell having only a right connection. The leftmost cell is the *communication cell*, which processes one input symbol at each time step. The local transition function is applied to each cell at the same time step. When the whole input is read, the end-of-input symbol # is processed. More details and results on iterative arrays may be found, e.g., in [3]. An IA is depicted in the following figure.

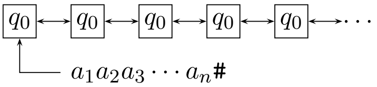


Fig. 1. An iterative array.

3 Space bounded iterative arrays

In the general model, along their computations, IAs may use as many cells as the input length. It is natural to investigate a sublinear cells usage. In analogy with the Turing machine model, we call space the amount of cells used by an IA. Formally, the space used in the computation $(x\#, c_0), \dots, (\varepsilon, c_{|x|+1})$ of a realtime-IA A on the word $x \in \Sigma^*$ is defined as

$$S(x) = \max\{i \in \mathbb{N} \mid c_{|x|+1}(i) \neq q_0\}.$$

The *strong space complexity* of A is the function $S : \mathbb{N} \to \mathbb{N}$ defined as

$$S(n) = \max\{S(x) \mid x \in \Sigma^* \text{ and } |x| = n\}.$$

Hence, $S(n)$ counts the maximum number of cells activated during the computations on words of length n. It is easy to see that, for any realtime-IA, $S(n) \leq n+1$. In this paper, we focus on *sublinearly* strongly space bounded realtime-IAs, i.e., with $S(n) = o(n)$. We denote by $\mathcal{L}_{rt}(S(n)\text{-IA})$ the class of languages accepted by $S(n)$ strongly space bounded realtime-IAs, i.e., the class of languages accepted by realtime-IAs which use in all the computations on inputs of length n *at most* $S(n)$ cells.

Examples

1. The language $\{a^m b^m c^m \mid m \geq 1\}$ can be accepted by a realtime-IA in $\log(n)$ strong space.

2. The language $\{a^{m^k} \mid m \geq 1\}$, for fixed $k \geq 2$, can be accepted by a realtime-IA in $\sqrt[k]{n}$ strong space. (Cf. [9])
3. The language $P = \{wcw^R \mid w \in \{a, b\}^*\}$ can be accepted by a realtime-IA in linear strong space.

An infinite proper space hierarchy

Let us now start to build an infinite proper hierarchy for sublinearly space bounded realtime-IAs. The first results are

1. REG $\subset \mathcal{L}_{rt}(\log(n)\text{-IA})$.
2. For every function $f(n) = o(n)$, $\mathcal{L}_{rt}(f(n)\text{-IA}) \subset \mathcal{L}_{rt}(\text{IA})$.

To go further in the hierarchy construction, we introduce the notion of ts-constructibility of functions by IAs (cf. [1]). This will enable us to get a general result leading to proper inclusions. Roughly speaking, the IA-ts-constructibility of a function f means that exactly at time steps $f(1), f(2), \ldots$ the communication cell enters an accepting state while not using more than $f^{-1}(n)$ cells for the computations on inputs of length n. It can be observed that the function $f(n) = n^k$, for any given integer $k \geq 2$, is IA-ts-constructible.

Lemma 1 *Let $f(n)$ be an IA-ts-constructible function which satisfies $f^{-1}(n) = \Omega(\log n)$, and let $g(n) = o(f^{-1}(n))$ be a non-decreasing function. Then, we obtain $\mathcal{L}_{rt}(g(n)\text{-IA}) \subset \mathcal{L}_{rt}(f^{-1}(n)\text{-IA})$.*

Corollary 1 *For any integer $k \geq 2$, it holds: $\mathcal{L}_{rt}(\log(n)\text{-IA}) \subset \mathcal{L}_{rt}(\sqrt[k]{n}\text{-IA})$ and $\mathcal{L}_{rt}(\sqrt[k+1]{n}\text{-IA}) \subset \mathcal{L}_{rt}(\sqrt[k]{n}\text{-IA})$.*

In conclusion, we obtain the following infinite space hierarchy:

Theorem 2 *For any integer $k \geq 2$,*

$$\text{REG} \subset \mathcal{L}_{rt}(\log(n)\text{-IA}) \subset \mathcal{L}_{rt}(\sqrt[k+1]{n}\text{-IA}) \subset \mathcal{L}_{rt}(\sqrt[k]{n}\text{-IA}) \subset \mathcal{L}_{rt}(\text{IA}).$$

Decidability questions

It is shown in [6] that almost all decidability questions for realtime-IAs are undecidable and not semidecidable. The same results can be extended to the restricted model of realtime-IAs working in $\log(n)$ strong space.

Theorem 3 *Emptiness, finiteness, infiniteness, universality, equivalence, inclusion, regularity, and context-freedom are not semidecidable for realtime-IAs working in $\log(n)$ strong space.*

Lower bounds for recognizing non-regular languages

According to Theorem 3, restricting to a logarithmic cell usage still leads to non-semidecidable questions. So, as a further restriction, we could consider realtime-IAs which are allowed to use only a *sublogarithmic* number of cells. Notice that an analogous investigation has been carried on even for space bounded Turing machines (see, e.g., [10, 11] for a survey on the sublogarithmic space world).

The next theorem shows that sublogarithmic space bounds reduce the computational capacity of realtime-IAs to the regular languages.

Theorem 4 *Let A be a realtime-*IA *$S(n)$ strongly space bounded. Then, either $S(n) \geq C \cdot \log(n)$, for some constant $C > 0$ and infinitely many n, or A accepts a regular language and the space used by A is bounded by a constant.*

We conclude by observing that the logarithmic space lower bound for non-regular language acceptance in Theorem 4 is *optimal.* It is enough, in fact, to consider the examples where a non-regular language is shown to be accepted by a realtime-IA in $\log(n)$ strong space.

References

[1] Buchholz, T., Klein, A., Kutrib, M.: Iterative arrays with small time bounds. In: Nielsen, M., Rovan, B. (Eds.): Mathematical Foundations of Computer Science (MFCS 2000), LNCS 1893, Springer-Verlag (2000) 243–252.

[2] Cole, S. N.: Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE Transactions Computers **C-18** (1969) 349–365.

[3] Kutrib, M.: Automata arrays and context-free languages. In: Where Mathematics, Computer Science and Biology Meet, Kluwer Academic Publishers (2001) 139–148.

[4] Kutrib, M., Malcher, A.: Fast iterative arrays with restricted inter-cell communication: constructions and decidability. In: Královič, R., Urzyczyn, P. (Eds.): Mathematical Foundations of Computer Science (MFCS 2006), LNCS 4162, Springer-Verlag (2006) 634–645.

[5] Kutrib, M., Malcher, A.: Real-time reversible iterative arrays. In: Csuhaj-Varjú, E., Ésik, Z. (Eds.): Fundamentals of Computation Theory (FCT 2007), LNCS 4639, Springer-Verlag (2007) 376–387.

[6] Malcher, A.: On the descriptional complexity of iterative arrays. IEICE Trans. Inf. Sci. **E87-D** (2004) 721–725.

[7] Malcher, A., Mereghetti, C., Palano, B.: Sublinearly space bounded iterative arrays. Technical Report 1/07, University of Frankfurt, 2007.

[8] Malcher, A., Mereghetti, C., Palano, B.: Sublinearly space bounded iterative arrays. Submitted to AFL 2008, 2008.

[9] Mazoyer, J., Terrier, V.: Signals in one-dimensional cellular automata. Theoret. Comp. Sci. **217** (1999) 53–80.

[10] Mereghetti, C.: The descriptional power of sublogarithmic resource bounded Turing machines. In: Geffert, V., Pighizzini, G. (Eds.): Descriptional Complexity of Formal Systems (DCFS 2007), Univ. P. J. Šafárik, Košice, Slovakia (2007) 12–26.

[11] Szepietowski, A.: Turing machines with sublogarithmic space. LNCS 843, Springer-Verlag, 1994.

[12] Umeo, H., Kamikawa, N.: Real-time generation of primes by a 1-bit communication cellular automaton. Fund. Inform. **58** (2003) 421–435.

Solving the problem of enforced restriction to few states while evolving cellular automata

Marcus Komann and Dietmar Fey

Institute for Computer Science, Friedrich-Schiller-University Jena, Germany
marcus.komann@googlemail.com fey@uni-jena.de

Abstract. Cellular automata are a promising model for many theoretical and real-life problems. The range of applications is enormous and the simplicity of description attracts scientists from various fields. However, finding the correct rules for a given problem often is a tedious task. One possible solution for this problem is evolving cellular automata. Unfortunately, using Genetic Algorithms presents a new barrier. The representation of rule sets as binary strings in Genetic Algorithms results in a search space for the evolution which increases exponentially in the number of states. But today's sophisticated problems mostly require state numbers which are large enough to cope with high complexity. This paper proposes a vision on cells not from a rule sets point of view but a view on small programs instead which are evolved much easier.

1 The problem: Complex tasks require many states

Research on dynamic systems like weather forecast, swarm behaviour, gas simulation, and uncounted others is one of the big issues of our time. Such systems and their models show huge effects even on smallest deviations and thus often reflect nature's laws much better than classic approaches. They can be used to integrate desired capabilities like self-organisation, self-healing, self-optimisation, and so on. And they depict the reality we are living in a much better way than classic models. Some of the mentioned problems strongly affect our everyday life and improving their characteristics is thus much desired.

However, describing such dynamic systems is often difficult. Analytic definitions and methods often fail due to the complexity of the systems and the inability of researchers to find simple modelling equations which can be solved in order to optimise certain parameters of the systems or to foresee results of events in present and future. Thus, other methods have to be used. One of them are Cellular Automata (CAs). Describing problems with CAs turns out to be much more simple because only the local interaction of the single cells has to be specified in contrary to other approaches where the behaviour of the complete system is directly modeled. Simulations then show if the local rules were chosen correctly and if the behaviour of the complete automata reflects the system it simulates or if the local rules have to be altered. Anyway, discussing the model of CAs is not the scope of this paper. For more details, please refer to [1], [2], or [3].

Using the CA model, writing down rules is plain simple. But the simplicity of the model does of course not simplify the problem itself. Finding the right local rules by hand is sometimes tedious if not impossible. Unexpected global effects often occur during simulation and pointing out the reasons in the local rules demand profound comprehension and experience as well with CAs as with the described problem itself. These requirements don't apply to every person using CAs and they slow the development processes. Therefore, evolution in the form of Genetic Algorithms was introduced to cellular automata. For some problems, evolving good local rules was effective. Please see [4], [5], or [6] for more information on the topic. But explaining all aspects of evolution is also not in the focus of this paper, so refer to the mentioned literature for details. We expect the reader to be familiar with the basic principles of evolution because we will only go into detail about the following important parts.

If using Genetic Algorithms, two things always have to be done very carefully: First, defining the quality function, and second, finding a suitable encoding of the individuals (the rule sets of CAs in this case). These two issues present the major hurdles for designing a successful evolutionary process and strongly influence its speed. Describing creation of the quality function in general is difficult because it is very problem-specific. For Genetic Algorithms and CA, the rule sets are usually encoded as binary strings. Thus, all states are encoded as binary numbers. A rule set then defines a cell's new state for each combination of neighbours' states. Saying it short, one string represents exactly one possible rule set for the given number of states and neighbours.

We wrote that the quality function strongly depends on the specific problem. But one statement about the encoding is always true: Having more states increases the length of the encoding strings and thus increases the number of possible rule sets exponentially. Consider a classic two-dimensional CA with s states and n neighbours. The amount of neighbours is depending on the kind of neighbourhood and the radius. For a NEWS neighbourhood with radius one, the amount is five because the cell itself also has to be counted here. For s states and n neighbours, we have s^n possible neighbourhood configurations for each cell. Thus, s^{s^n} different rule sets are possible making the growth exponential in s.

For larger numbers of states, it is obviously not possible to simulate all possible rule sets in order to retrieve their quality. For example, having six states and NEWS neighbourhood results in more than 221×10^{20} possible rule sets. The idea of the genetic process now is to merge different rule sets or slightly alter single rule sets that qualified as good in order to decrease the number that has to be simulated for finding a suitable rule set for a given problem. All possible rule sets thus build the search space the genetic process is searching in. Summing up, large numbers of states result in huge search spaces and are therefore likely to result in poor quality of the found CA or in very long evolution time.

As stated before, many complex systems rely on the model of CAs. But describing problems with few states like four or less is difficult if not impossible in most cases. The need for higher numbers of states results from the need for the description of different weather conditions, several kinds of animals or animal states/behaviours or particles, not mentioning specific boundary cases. Even if one is able to find a minimal state set for a given problem or is otherwise able to reduce the number of applicable rule sets, the amount of states required might exceed reasonable sizes of the search space by far. In the end, we have to find a different representation of the problem in order to come to acceptable results in reasonable times using evolution.

2 A solution: Evolving small programs instead of rule sets

For a fixed issue, faster computers or evolution directly in hardware [7], [8] decrease the time until a solution is found. But these approaches can't cope with the root of the problem. They are only reducing computation time by a constant factor. This might work out in specific cases, but not in the big scheme. We have to find a model where the search space is not exponentially dependent on the number of states.

The solution is to look at single cells of the CA not with a view on states but to consider them as small processors capable of executing small programs instead. They still need to have states though but it is possible to decrease the required number of these states significantly compared to the formerly mentioned approach. Apart from that, this model fits very well to real-world, fine-grained, massively-parallel architectures (see Sect. 3.2) for details.

Evolving programs is called *Genetic Programming* and is strongly connected to the works of John Koza [9]. He proposed to take a set of elementary operations (functions and literals), put them together arbitrarily, and to let evolution form programs from this primordial soup which suit a given problem. This kind of evolution differs not much from Genetic Algorithms in principle. It also uses mutation, crossover, and selection to increase the fitness of individuals. The difference lies in the representation of individuals and the quality function.

The quality function now is responsible for checking the strength of evolved programs. But the basic idea of quality doesn't change and neither do quality criteria of a specific problem. Transforming the quality function from Genetic Algorithms to Genetic Programming is thus mostly relatively easy. Apart from basic “desired output" measurement, the quality function has to be extended by some factors. First to mention is a weighing factor for the length of the program. If one does not attach it to the quality function, the evolutionary process will tend to create very long, super-specialised programs which are on one hand time- and maybe hardware-cost-expensive and which don't generalise well on the other hand. A second weighing factor may be the real time a program needs for execution. For example, a multiplication by two with a multiplying unit might cost the same program size but twice the real execution time of a left shift.

The larger difference lies in the representation of individuals. Using Genetic Algorithms, we had bit strings which encode the states and the state transitions. Using Genetic Programming, we now have small programs which are executed by each cell. The cell model therefore has to be able to cope with such instructions, extending the classic model. The set of basic instructions a cell can execute build the core from which evolution derives programs. Not every instruction has to be used in optimal programs though. For example, making an edge detection of objects does not necessarily require arithmetic functionality of the cells or an agent moving from left to right does not require an instruction to go north or south.

The evolutionary process now again merges and alters good quality individuals.The final program can be viewed as a binary tree. The evolutionary process cuts off arbitrary partial subtrees of two individuals each and exchanges them mutually creating two new individuals (crossover). Or it simply exchanges one instruction with another one from the core set (mutation). See Fig. 1 and Fig. 2 for illustration. This procedure might create programs which are not allowed. The scientist has to take care that either all possible combinations of literals and functions are valid constructs or has to take care of eliminating such malicious individuals.

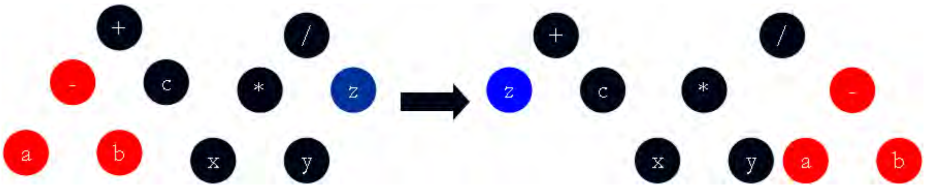


Fig. 1. Genetic Programming crossover: Nodes z and - and their subtrees are mutually exchanged (z does not have a subtree).

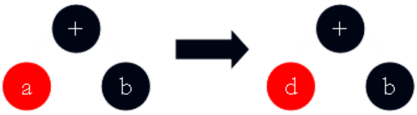


Fig. 2. Genetic Programming mutation: Node a is mutated to d.

As already told, this view on CA evolution of course requires leaving the classic CA model. But it makes it possible to exploit the capabilities of processors in the cells of a cellular automaton extending the set of possible applications which are easy to describe. Another point is that such program-CAs can often be re-transformed into classic CAs by emulating the instructions with states, for example when thinking of agents moving on the CA and mimicking the movement directed by instructions with states. This would of course mostly require a lot of states and vastly increase the complexity.

3 An example: Machine Vision with massively-parallel embedded cameras

Having proposed the new view on CA evolution, we present a real problem scenario in this section which originates from production industry. The problem is specified in detail and the so called *Marching Pixels* project is presented. This project is part of the priority program *Organic Computing* of the German Research Foundation. In the end, we show that the presented approach is capable of evolving suitable algorithms for this industrial vision problem.

3.1 The problem: Fast detection of objects in binary images

In modern factories, machine vision is an elementary part of the production processes. Robots have to see in order to grab, drill, grind, or in general handle tools and work pieces. With increasing production speeds, the need for extremely fast vision systems is ever-growing. On the other hand, it is also important that these vision systems don't become too large for reasons of power consumption, price, or simply limited space for installation, e.g. on a robot's gripper arm.

One of these vision problems is the detection of objects and their attributes in binary images. Detection means classification of objects, for example lying on an assembly line, out of a set of known objects. It also means detection of defective or incomplete work pieces which the complete production system has to sort out. Apart from the object classification, it is also important to detect some properties of the single objects like size, edge lengths, rotation, and centroids.

When these opposing aims of low size and high speed meet, classic architectures reach their limits. This is especially true if the systems don't have to detect one but many objects at a time whose number is not known in advance. A practical example for this are small pieces like nails or screw-nuts lying on a fast moving assembly line.

3.2 The Marching Pixels approach

One system architecture that is able to cope with these opposing aims is developed in the Marching Pixels project. This architecture is presented here shortly to give the reader an idea of it and because the evolution presented in Sect. 3.3 is based on this architecture. For more details on the system, the algorithms, capabilities and weaknesses, and implementations in VHDL and FPGA, please refer to earlier publications like [10], [11] or [12].

Put very short, the basic idea of Marching Pixels is to take a binary image, load it into a massively-parallel field of simple processing elements, and to let agents, so called Marching Pixels (MPs), travel around this field in order to detect objects and their attributes. These agents thereby exploit emergence [13] to fulfill their goals. The advantage of this approach is that the complete architecture can be implemented on a relatively small dedicated chip and thus in a small embedded vision system (including camera and input/output), and that the massive parallelism grants large compute power which can be used to meet strict real-time requirements.

This description divides an MP vision system into two parts. The first one is an architectural layer where the field of processing elements in a whole, the processing elements themselves in detail, and the input/output characteristics have to be defined. On the upper layer, the algorithms executed on the single processing elements have to be found and tested. This is where the Marching Pixels run. It is also the layer which is close to CAs because the steering of the Marching Pixels can be (and has been) done by means of cellular automata.

Architectural layer In this layer, the underlying hardware is described. Figure 3 shows the principle. On the left side, the processor element (PE) array on a chip can be seen. The PEs have a specific local neighbourhood which can be defined freely. In the figure, the PEs have NEWS neighbourhood. PEs don't have connectivity across the borders. Outer PEs are used for outside communication instead.

On the right side of the figure, you can see a PE in detail. It has an optical detector on top which captures a part of the image, i.e. one pixel in the most simple case. This pixel is then binarised and used as input to digital processing. In the digital part, arbitrary digital operations can be used like binary logic, shift operations, multiplexing, or even ALUs. Apart from that, it is possible to use Flip-Flops as local memory for each single PE. Using CAs, the state machine is modeled here in digital hardware and the states of the cells are stored in local memory.

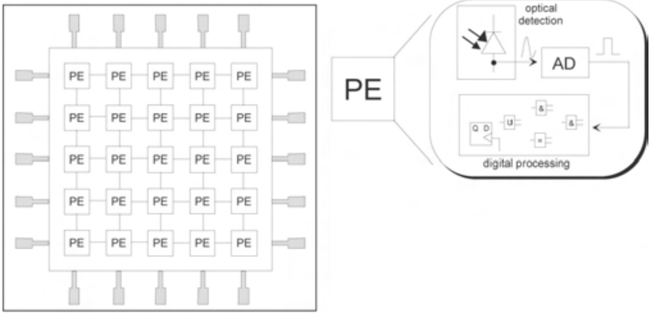


Fig. 3. Architecture of processor element array and of single PE.

Algorithmic layer Having PEs as CA-cell-alike units with the extraordinary ability of executing digital logic, we now can think of algorithms which solve the previously described problem of fast object detection in binary images. In the Marching Pixels project, we oriented on ant-like behaviour to steer agents across the PE array (and thus the image) with the goal of visiting certain pixels and compressing the information. In the end, the centroid pixel shall be found and the desired information like size, rotation, or edge lengths shall be gathered.

On their way, these agents aka Marching Pixels (MPs) can mutually interact directly or via indirect, so-called *stigmergic*, communication. MPs can be born, can unite, and they can die. Exploiting these capabilities, we are able to create several emergent algorithms with different capabilities and requirements. For example, among others we have algorithms which detect rectangles, one which detects convex objects, and one which detects concave objects. The state machine and memory (and thus hardware) requirements increase from one to the next along their higher functionality. We won't go into detail about the specific algorithms here. Please refer to the literature mentioned at the beginning of Sect. 3.2 for details.

3.3 Evolutionary results

If we sum up shortly, we have the problem of detection of objects in binary images, we have a hardware structure with parallel processors that emulate the image, and we have algorithms which steer agents around this CA-like field in order to visit all or specific pixels. One of the major problems in this complete system is finding algorithms suitable for specific problems. We don't know if the algorithms we already developed by hand are the best or even good ones. Evolution of such algorithms is thus a promising idea. Apart from that, a designer might not want to cope with the complete design process when parameters change. A ready evolution system helps him to get results relatively fast without much of work or brainpower.

Genetic Algorithms The results of our work on Genetic Algorithms can be found in detail in [14] and shall be presented here very shortly. We evolved uniform as well as non-uniform CAs in order to solve problems like shrinking objects to their distinct centroid pixel. Rule tables were encoded as binary strings. Quality of a rule table was measured by applying evolved rule tables to input images with objects and comparing the final output of the CA computation with desired output images (see Fig. 4).

The quality (fitness) function thereby consisted of two parts. The first part counted wrong pixels resp. states after execution of the CA compared to the desired goal images. The second part took care for weighing the distance of wrong pixels to their correct places. The idea is that rule tables get higher fitness ratings if correct states are closer to their correct places making it easier for the evolutionary process to find better results.

The results of evolving CAs for this application were rather disappointing. Evolved rule tables were sometimes able to cope with their "learned" images

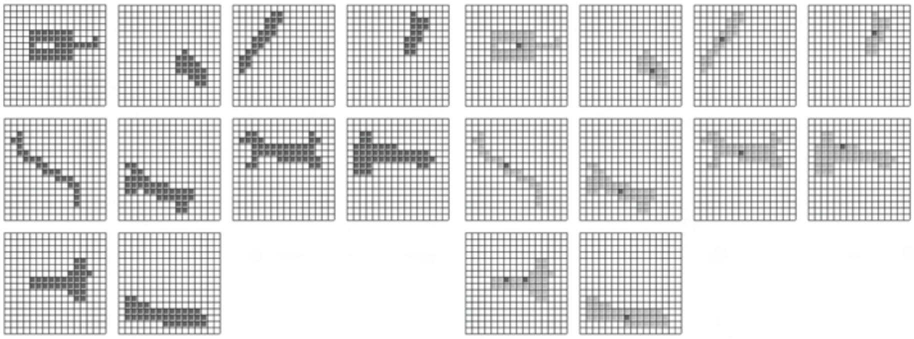


Fig. 4. Left: Simple input images of tools; Right: Calculated output images (Grey pixels represent the objects and are not present in the original output image. They can be seen here just for orientation).

but robustness to small deviations like image noise, smallest rotations or scaling of the objects was not given. Not to mention universality concerning unlearned objects.

Non-uniformity in the sense of Sipper [6] where the rules of single cells may differ in one CA also did not achieve better fitnesses. This was mainly due to the locality of the problem and the indeterministic appearance positions of objects. If the position was not fixed in advance, non-uniform CAs converged to uniform ones.

The biggest reason for these bad results was the forced limitation to few states. We are convinced that a larger number of states of the CA cells would have yielded better results. But the evolution time increases exponentially when raising the number of states. This held us off giving the evolutionary process more than three states to work with.

Genetic Programming We already proposed using Genetic Programming instead of Genetic Algorithms in Sect. 2. The Marching Pixels architecture described in Sect. 3.2 gives us not only a state machine but also memory and logic functionality. This enables us to model the agents' walk not only by means of CA rule tables but to implement real agent behaviour with decisions and calculations. With Genetic Programming, we have an evolutionary tool that exploits this possibility because it models agents well.

As described in Sect. 2, the individuals are now encoded as a set of instructions. The token list, from which evolution choses in the tests described here consisted of `IfFoodAhead`, `Prog2`, `Noop`, `TurnLeft`, `TurnRight`, and `StepForward`. A pixel which the agent should visit is denoted as *food*. `Prog2` simply executes the two following instructions and is required to spread the binary program tree. Turning and stepping forward is self-explaining. `Noop` is required because the `IfFoodAhead` instruction executes either the true or the false branch. Programs might need the if-branch but not the if-not-branch resulting in a `Noop` as second argument for `IfFoodAhead`.

Quality of individuals of the evolutionary population resp. the programs is evaluated relatively similar to Genetic Algorithms. Again, input images are given, then the processing takes place, and afterwards, the deviation of the processed image to the desired goal image is evaluated. The difference is that the processing is now not a number of steps according to a CA rule table but an execution of the programs instead. The second difference is that the goal image is not a specific pixel but consists of the pixels which should have been visited by an optimal agent. In this case, the agent should visit all object pixels. On its way, the agent accumulates coordinates of the visited object pixels. The sum of the coordinates and the number of visited pixels is later used for calculating the centroid of the object. This capability is complex and thus given to the agent a priori instead of also being evolved. The task of the evolutionary process is to find a proper marching scheme for the agent.

Besides pure effectivity, another factor has to be included into the quality function. It is a penalty increasing with the number of instructions. It this is not added, the evolutionary process tends to create very long programs specialising very much on the training sets. This is undesired because then execution takes too long and the costs for implementation in hardware rise.

At first, we used squares and rectangles as test objects for the Genetic Programming evolution. The starting point of the agent was given with the upper left corner pixel of the object. For a human developer, the program an agent should execute to visit all pixels of a square or a rectangle is obvious. It should be something like either run clockwise or counterclockwise along the edges and then towards the center directly beneath the already visited pixels forming some spiral until all pixels are visited. Or it should be something like going right in the first row until the right edge is found, then go south to the second row and walk leftwards until the left edge is found; followed again by a step southwards and the beginning from start. The idea was to check if the evolutionary process is able to find such a program. The given set of instruction makes it possible to find such an algorithm.

Indeed, such programs were found. From 23 tests, 20 used exactly the described running procedure (see Fig. 5 left). They sometimes differed in the way the agent turned around at corners but the principle was equal. The best program evolved was `IfFoodAhead StepForward TurnLeft` with an agent starting southwards in the upper left corner of the object. Another example, `IfFoodAhead StepForward TurnRight` is slower at the corners if it starts there and runs southwards. Two programs let the agents leave the objects at corners returning into the object afterwards and then running along the edges (see Fig. 5 middle). And one program used an unforeseen marching scheme which was also successful. The agent there ran straight until meeting a point where it has to turn. But different to the other programs, it did not turn 90 degrees, but sometimes also 180 degrees after a sidewards step (see Fig. 5 right). Its program is `Prog2 Prog2 Noop IfFoodAhead StepForward TurnLeft StepForward`. The corresponding program tree is shown in Fig. 6. It can be seen that the Noop leaf as well as its direct mother node Prog2 are useless and should be deleted. The IfFoodAhead branch should be the direct left successor of the root. Anyhow, this is the program evolved which survived as the fittest. None of the programs failed totally. The smallest programs required just three instructions while the longest ones needed nine.

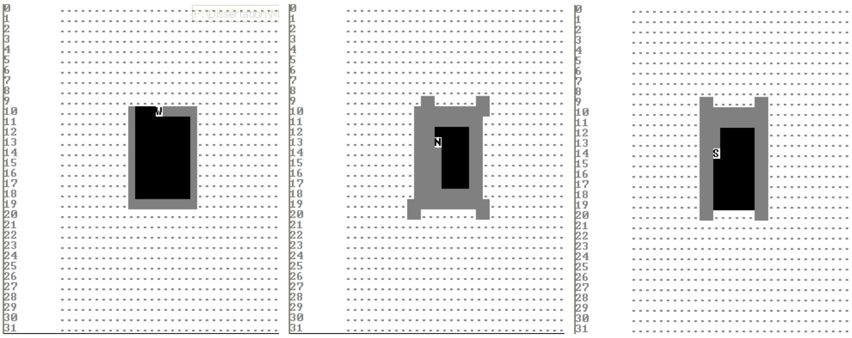


Fig. 5. Left: Expected and achieved result; Middle: Two different running examples; Right: Single totally unexpected result. Black pixels belong to the object, white pixels belong to the background, grey pixels have already been visited by the agent.

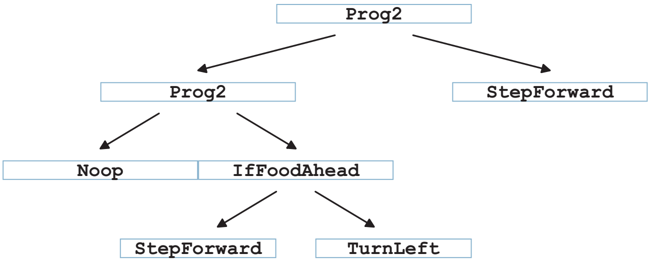


Fig. 6. Program tree of the single extraordinary result of the evolution.

For a second test set, we used the tools seen in Fig. 4. At first, single tools were used for teaching and it was checked if suitable programs are found. Some of the programs detected their objects quite well although they mostly missed some single food pixels. But a rate of about 95 percent found food for the best individuals is satisfactory. The negative aspect of the programs was their length. The best ones had about 50 instructions while the worst ones had some hundreds. This is of course not desired and has to be worked on. From a robustness view, the programs still worked if the tools were scaled or rotated or if noise disturbed the image. Fig. 7 shows an exemplary test on a bench vice. The agent starts in the upper left corner of the image and at first eats the noise pixels at the left side of the vice. It then enters the vice and visits all object pixels hardly leaving the object.

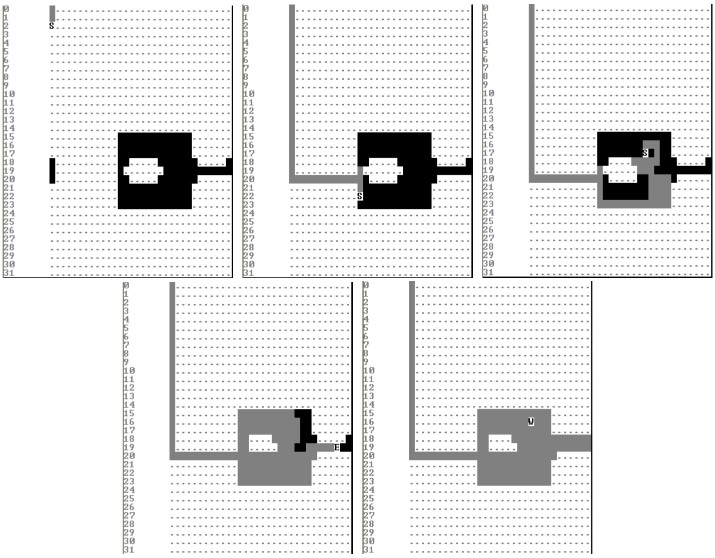


Fig. 7. Simulation of an agent's detection of a bench vice.

Then, more than one tool image was used for teaching and it was checked if the taught tools were detected correctly. The results are largely the same as for single objects. Some programs failed totally but the best ones did their job well. The problem with the program sizes also persisted. Apart from that, it showed that different objects whose shapes are similar had better results although differently-shaped objects ended up with some good results, too.

At last, the best former programs were applied to untaught images. The result was promising. Some of the algorithms were able to detect unlearned objects creating some kind of universality. This was again easy if learned objects and test objects looked similar. That is a somehow natural and expected result. But also the robustness against totally different objects was good for some programs. And these were no single exceptions. About half of the programs did satisfactory while about one quarter did really well.

These were some quite positive results. Especially the robustness and the universality aspects seem to be promising. But more tests have to be made with larger training sets and more untaught challenging images. Furthermore, the length of the evolved programs is not yet satisfying and should be reduced by increasing the penalty for a large amount of instructions.

The tests also showed, that it would be useful to enhance the general fitness factors with some other weighing factors. In future, we are going to include a penalty factor if agents leave the object pixels because it on one hand causes higher execution time and on the other hand might result in interferences of agents originated from different objects. It is also possible to decrease the fitness of programs whose agents cross pixels more than once in order to decrease execution time. The weight of each of these additional factors has to be adjusted cautiously. They will otherwise alter the effectivity of the evolution.

4 Summary

In this paper, we proposed a new vision on massively-parallel models. The cellular automaton has been and still is very successful in some research areas. The problem is just that it is sometimes difficult to find the correct rules for a given problem. Evolution has helped a lot in solving this problem and is thus important for the CA community. However, it is sometimes difficult to evolve rules especially for complex systems because the search space for a Genetic Algorithm grows exponentially in the number of states.

An option to solve this problem is the proposed shift of view on CAs. If we allow ourselves to extend the cell model by logic functionality and memory, we have a much more powerful tool to describe such complex systems. Another benefit of such a shift is that it is much easier to evolve programs instead of rule sets using Genetic Programming.

We showed the feasibility of this approach for the example of embedded machine vision. Some programs have been successfully evolved for differently sophisticated object detection problems as a test of concept. Some of the evolved programs worked fine concerning the detection. The biggest problem was that the number of instructions became too large for the more complex problem of tool detection. But this might be solved by adjusting the quality function and is part of future work.

References

[1] John Von Neumann: Theory of Self-Reproducing Automata. University of Illinois Press, Champaign, IL, USA, 1966.

[2] Tommaso Toffoli and Norman Margolus: Cellular Automata Machines: A New Environment for Modeling. MIT Press, Cambridge, MA, USA, 1987.

[3] Stephen Wolfram: A New Kind of Science. Wolfram Media Inc., Champaign, Ilinois, US, United States, 2002.

[4] N. H. Packard: Adaptation toward the edge of chaos. In M. F. Shlesinger J. A. S. Kelso, A. J. Mandell, editor, Dynamic Patterns in Complex Systems, pages 293-301, Singapore, 1988. World Scientific.

[5] Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield: Revisiting the edge of chaos: Evolving cellular automata to perform computations. Technical Report Santa Fe Institute Working Paper 93-03-014, 1993.

[6] Moshe Sipper: Evolution of Parallel Cellular Machines: The Cellular Programming Approach. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[7] Mathias Halbach, Rolf Hoffmann, and Lars Both: Optimal 6-state algorithms for the behavior of several moving creatures. In Lecture Notes in Computer Science, volume 4173/2006 of Cellular Automata, pages 571-581, Berlin / Heidelberg, 2006. Springer.

[8] Wolfgang Heenes, Rolf Hoffmann, and Sebastian Kanthak: Fpga implementations of the massively parallel gca model. In IPDPS í05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPSí05) - Workshop 14, page 262.2, Washington, DC, USA, 2005. IEEE Computer Society.

[9] John R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, 1992.

[10] Dietmar Fey and Daniel Schmidt: Marching-pixels: A new organic computing paradigm for smart sensor processor arrays. In CF í05: Proceedings of the 2nd conference on Computing frontiers, pages 1-9, New York, NY, USA, 2005. ACM.

[11] Marcus Komann and Dietmar Fey: Realising emergent image preprocessing tasks in cellular-automaton-alike massively parallel hardware. International Journal of Parallel, Emergent and Distributed Systems, 22(2):79-89, 2007.

[12] Dietmar Fey, Marcus Komann, Frank Schurz, and Andreas Loos: An Organic Computing architecture for visual microprocessors based on Marching Pixels. In ISCAS, pages 2686-2689. IEEE, 2007.

[13] Tom De Wolf and Tom Holvoet: Emergence versus self-organisation: Different concepts but promising when combined. In Sven Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, editors, Engineering Self-Organising Systems, volume 3464 of Lecture Notes in Computer Science, pages 1-15. Springer, 2004.

[14] Marcus Komann, Andreas Mainka, and Dietmar Fey: Comparison of evolving uniform, non-uniform cellular automaton, and genetic programming for centroid detection with hardware agents. In Victor E. Malyshkin, editor, PaCT, volume 4671 of Lecture Notes in Computer Science, pages 432-441. Springer, 2007.

Analysis of five-cell neighborhood linear cellular automaton

Masaya Nohmi

Faculty of computer science and systems engineering, Kyushu Institute of Technology,
Iizuka, 820-8502, Japan
nohmi@ai.kyutech.ac.jp

Abstract. Properties of a five-cell neighborhood linear cellular automaton are investigated. From the eigenpolynomial of the global state transition matrix, properties of a linear cellular automaton, such as the structure of the global state transition diagram, the longest transient length and the longest period length, are determined. The method is useful for other linear cellular automata with some arrangements.

1 Introduction

Many versions of cellular automata (CA) with three neighborhood local state transition rule are known. In [1], algebraic properties of CA with local state transition rule number 90 are investigated. In [2] and [3], properties of CA's with cyclic boundary condition are investigated, using a polynomial expression. In [4], properties of CA's with fixed value boundary condition are investigated, using algebraic integers. These studies mainly deal with three neighborhood CA's. But the behaviors of CA's with five or more neighbors are full of variety [5]. In this article, properties of a linear cellular automata (LCA) with five-cell neighborhood local state transition rule are investigated.

2 Definitions

Let LCA-$(0,1;0;1,1)_n$ be a linear cellular automaton (LCA) defined by the following rules. The n cells are arranged linearly, and each cell takes a value 0 or 1. The state of i-th cell at a time step t is denoted by

$$x_i^{(t)} \in \mathbf{Z_2} = \{0,1\}, \tag{1}$$

and the global configuration at a time step t is denoted by the notation

$$x^{(t)} = {}^t(x_1^{(t)}, x_2^{(t)}, \cdots, x_n^{(t)}) \in \mathbf{Z}_2^n, \tag{2}$$

where the notation ${}^t(x_1^{(t)}, x_2^{(t)}, \cdots, x_n^{(t)})$ means the transposed vector of $(x_1^{(t)}, x_2^{(t)},$ $\cdots, x_n^{(t)})$. The local state transition map is

$$\lambda(z_{-2}, z_{-1}, z_0, z_1, z_2) = z_{-1} + z_1 + z_2 \in \mathbf{Z}_2, \tag{3}$$

where the sum $z_{-1} + z_1 + z_2$ is calculated in the finite field $\mathbf{Z}_2$. The local state transition is defined by the equation

$$x_i^{(t+1)} = \lambda(x_{i-2}^{(t)}, x_{i-1}^{(t)}, x_i^{(t)}, x_{i+1}^{(t)}, x_{i+2}^{(t)}), \tag{4}$$

where the boundary conditions $x_{-1}^{(t)} = x_0^{(t)} = x_{n+1}^{(t)} = x_{n+2}^{(t)} = 0$ are assumed. That is, the fixed value zero boundary conditions are assumed. The global state transition map of LCA-$(0,1;0;1,1)_n$ for configurations of n cells is denoted by the notation

$$\tau_n \colon \mathbf{Z}_2^n \to \mathbf{Z}_2^n, \qquad \tau_n(x^{(t)}) = x^{(t+1)}. \tag{5}$$

For example, a configuration of LCA-$(0,1;0;1,1)_6$

$$x^{(t)} = {}^t(0,0,1,0,1,1) \tag{6}$$

changes into the configuration

$$\tau_n(x^{(t)}) = x^{(t+1)} = {}^t(1,1,1,1,1,1) \tag{7}$$

at the next time step. The representation matrix of τ_n on the standard basis is

$$A_n = \begin{pmatrix} 0&1&1&&&&&&\\ 1&0&1&1&&&&&\\ 0&1&0&1&&&&&\\ &0&1&0&&&&&\\ &&&&\ddots&&&&\\ &&&&&0&1&1&\\ &&&&&1&0&1&1\\ &&&&&0&1&0&1\\ &&&&&&0&1&0 \end{pmatrix} \in M_n(\mathbf{Z}_2). \tag{8}$$

Figure 1 shows the global state transition diagram of LCA-$(0,1;0;1,1)_n$ in the case $n = 4$, where column vectors are replaced to row vectors.

Figure 2 shows the global state transition diagram of LCA-$(0,1;0;1,1)_n$ in the case $n = 6$, where configurations are abbreviated to small circles.

Let $e_1 = {}^t(1,0,\cdots,0) \in \mathbf{Z}_2^n$ be a standard base of $\mathbf{Z}_2^n$, and let

$$e_i = A^{i-1}e_1, \qquad (2 \le i \le n). \tag{9}$$

Proposition 2.1. *The vectors e_1, e_2, $\cdots$, e_n are a basis of $\mathbf{Z}_2^n$.*

Proof. Let

$$e_i = {}^t(e_{i,1}, \cdots, e_{i,n}), \qquad (1 \le i \le n),$$
$$e_{i,j} \in \mathbf{Z}_2, \qquad (1 \le i,j \le n).$$

The following equations hold:

$$e_{i,i} = 1, \qquad (1 \le i \le n)$$
$$e_{i,j} = 0, \qquad (1 \le i < j \le n).$$

Therefore, the vectors e_1, e_2, $\cdots$, e_n are linearly independent in $\mathbf{Z}_2^n$.

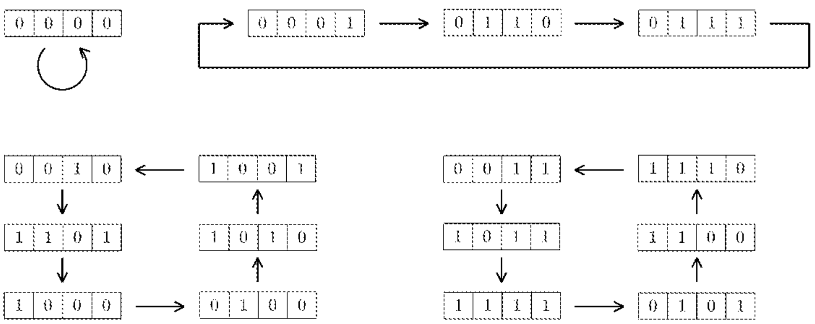


Fig. 1. The global state transition diagram of LCA-$(0,1;0;1,1)_4$.

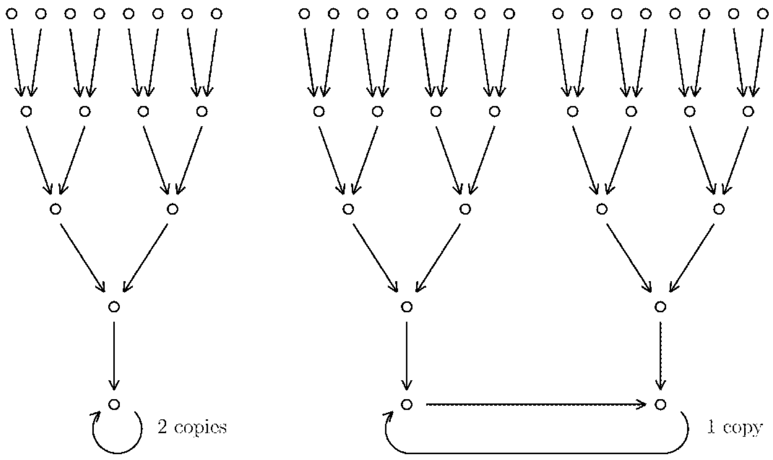


Fig. 2. The global state transition diagram of LCA-$(0,1;0;1,1)_6$.

The representation matrix of τ_n on the basis $e_1,\, e_2,\, \cdots,\, e_n$ is

$$A'_n = \begin{pmatrix} 0 & & & & & & a_0 \\ 1 & 0 & & & & & a_1 \\ & 1 & 0 & & & & \\ & & 1 & \ddots & & & \vdots \\ & & & \ddots & 0 & & \\ & & & & 1 & 0 & a_{n-2} \\ & & & & & 1 & a_{n-1} \end{pmatrix} \in M_n(\mathbf{Z}_2), \tag{10}$$

where the eigenpolynomial of A_n is

$$\begin{aligned} &\varphi_n(t) = t^n + a_{n-1}t^{n-1} + a_{n-2}t^{n-2} + \cdots + a_1 t + a_0 \in \mathbf{Z}_2[t], \\ &a_i \in \mathbf{Z}_2, \qquad (0 \le i \le n-1). \end{aligned}$$

Because a configuration $x \in \mathbf{Z}_2$ of LCA-$(0,1;0;1,1)_n$ is uniquely represented as

$$\begin{aligned} &x = x_1 e_1 + x_2 e_2 + \cdots + x_n e_n, \\ &x_i \in \mathbf{Z}_2, \qquad (1 \le i \le n), \end{aligned}$$

it corresponds to a polynomial

$$f(t) = x_1 + x_2 t + \cdots + x_n t^{n-1} \in \mathbf{Z}_2[t] \tag{11}$$

uniquely under the condition that $\deg(f(t)) \le n-1$ and $f(A_n)e_1 = x$. Inversely, a polynomial $f(t) \in \mathbf{Z}_2[t]$ corresponds to a configuration of LCA-$(0,1;0;1,1)_n$ by the mapping

$$\begin{aligned} &\Pi_n :\ \mathbf{Z}_2[t] \to \mathbf{Z}_2^n, \\ &\Pi_n(f(t)) = f(A_n)e_1. \end{aligned} \tag{12}$$

A polynomial $f(t) \in \mathbf{Z}_2[t]$ is uniquely represented as

$$\begin{aligned} &f(t) = g(t) + h(t)\varphi_n(t), \\ &g(t), h(t) \in \mathbf{Z}_2[t], \quad \deg(g(t)) < n. \end{aligned} \tag{13}$$

The kernel of Π_n is the principal ideal

$$(\varphi_n(t)) = \{h(t)\varphi_n(t) \mid h(t) \in \mathbf{Z}_2[t]\}, \tag{14}$$

because $\varphi_n(A_n) = O$ from Cayley-Hamilton's theorem. Therefore, a configuration of LCA-$(0,1;0;1,1)_n$ uniquely corresponds to a residue class in

$$P_n[t] = \mathbf{Z}_2[t]/(\varphi_n(t)). \tag{15}$$

We call each element of $P_n[t]$ a polynomial representation of a configuration of LCA-$(0,1;0;1,1)_n$.

Proposition 2.2. *Let $f(t)$ be the polynomial representation of a configuration $x^{(t)}$ of LCA-$(0,1;0;1,1)_n$. The polynomial representation of the next time step configuration $x^{(t+1)}$ is $tf(t)$.*

Proof. Let

$$x^{(t)} = x_1e_1 + x_2e_2 + \cdots + x_ne_n, \qquad x_i \in \mathbf{Z}_2 \quad (1 \le i \le n)$$

be a configuration of LCA-$(0,1;0;1,1)_n$. The polynomial representation of $x^{(t)}$ is

$$f(t) = x_1 + x_2t + \cdots + x_nt^{n-1} \in \mathbf{Z}_2[t].$$

The configuration at the next time step is

$$\begin{aligned} x^{(t+1)} &= x_1A'_ne_1 + x_2A'_ne_2 + \cdots + x_nA'_ne_n \\ &= x_1e_2 + x_2e_3 + \cdots + x_{n-1}e_n \\ &\qquad +x_n(a_0e_1 + a_1e_2 + \cdots + a_{n-1}e_n), \end{aligned}$$

where the eigenpolynomial of A_n is

$$\varphi_n(t) = t^n + a_{n-1}t^{n-1} + a_{n-2}t^{n-2} + \cdots + a_1t + a_0 \in \mathbf{Z}_2[t].$$

The polynomial representation of $x^{(t+1)}$ is

$$x_1t + x_2t^2 + \cdots + x_{n-1}t^{n-1} + x_n(a_0 + a_1t + \cdots + a_{n-1}t^{n-1}) \in \mathbf{Z}_2[t].$$

This is equal to

$$x_1t + x_2t^2 + \cdots + x_{n-1}t^{n-1} + x_nt^n = tf(t),$$

because $\varphi_n(t) = 0$ in the equivalence class $P_n[t]$.

3 Analysis of LCA

Let

$$d_n = \max\{\, i \in \mathbf{Z} : t^i \mid \varphi_n(t) \,\}, \tag{16}$$

where the notation $t^i \mid \varphi_n(t)$ means that the eigenpolynomial $\varphi_n(t)$ is exactly divisible by t^i.

Assume that $\varphi_n(t) \ne t^n$. The linear operation of A_n restricted on the subspace spanned by the vectors

$$e_i, \qquad (d_n + 1 \le i \le n) \tag{17}$$

is regular. Therefore, the configurations

$$t^i, \qquad (d_n \le i \le n-1)$$

are cyclic configurations of LCA-$(0,1;0;1,1)_n$. Let

$$L_n[t] = \begin{cases} \left\{ \sum_{i=d_n+1}^{n} x_i t^{i-1} \;\middle|\; x_i \in \mathbf{Z}_2 \right\}, & (\varphi_n(t) \neq t^n) \\ \{0\}, & (\varphi_n(t) = t^n) \end{cases}. \tag{18}$$

Any configuration $f(t) \in L_n[t]$ is cyclic.

Assume that $d_n \geq 1$. Because the configuration t^{d_n} is cyclic, it has predecessor configurations. Let

$$\tau^{-i}(t^{d_n}), \qquad (i \geq 1)$$

be the predecessor configuration of t^{d_n} uniquely determined under the condition that $\tau^{-i}(t^{d_n}) \in L_n[t]$ and $\tau^i(\tau^{-i}(t^{d_n})) = t^{d_n}$. Let

$$u_i = \begin{cases} t^{d_n - i} - \tau^{-i}(t^{d_n}), & (1 \leq i \leq d_n) \\ 0 & (i = 0) \end{cases}. \tag{19}$$

The state transition of u_i is

$$\tau_n(u_i) = tu_i = u_{i-1}, \qquad (1 \leq i \leq d_n). \tag{20}$$

The set

$$W(0,i) = \{f(t) \in P_n[t] \mid t^i f(t) = 0\}, \qquad (0 \leq i \leq d_n) \tag{21}$$

is a subspace of $P_n[t]$. From the definition of u_i,

$$u_i \in W(0,i) - W(0,i-1). \tag{22}$$

Let

$$U_n[t] = \begin{cases} \left\{ \sum_{i=1}^{d_n} x_i u_i \;\middle|\; x_i \in \mathbf{Z}_2 \right\}, & (d_n \geq 1) \\ \{0\}, & (d_n = 0) \end{cases}. \tag{23}$$

Theorem 3.1. *Any configuration $f(t)$ of LCA-$(0,1;0;1,1)_n$ is uniquely represented as*

$$f(t) = u(t) + l(t), \qquad u(t) \in U_n[t], \quad l(t) \in L_n[t].$$

That is,

$$P_n[t] \cong U_n[t] \oplus L_n[t].$$

Proof. Let

$$f(t) = x_1 + x_2 t + \cdots + x_n t^{n-1}$$

be a configuration of LCA-$(0,1;0;1,1)_n$. From the definition of u_i,

$$f(t) = \sum_{i=1}^{d_n} x_{d_n - i+1} u_i + \sum_{i=1}^{d_n} x_{d_n-i+1} \tau^{-i}(t^{d_i}) + \sum_{i=d_n+1}^{n} x_i t^{i-1}.$$

Because

$$\sum_{i=1}^{d_n} x_{d_n-i+1}u_i \in U_n[t],$$

$$\sum_{i=1}^{d_n} x_{d_n-i+1}\tau^{-i}(t^{d_i}) + \sum_{i=d_n+1}^{n} x_i t^{i-1} \in L_n[t],$$

any configuration of LCA-$(0,1;0;1,1)_n$ can be represented as an element of $U_n[t] + L_n[t]$. The sets $U_n[t]$, $L_n[t]$ are subspace of $P_n[t]$, and $U_n[t] \cap L_n[t] = 0$. Therefore, $P_n[t] \cong U_n[t] \oplus L_n[t]$.

Let G_n be a directed graph defined by the following rules. The set of all vertices of G_n is

$$V(G_n) = \{(u(t), l(t)) \,|\, u(t) \in U_n[t], l(t) \in L_n[t]\}. \tag{24}$$

Each vertex of G_n is adjacent to exactly one edge as an initial vertex. For each initial vertex of G_n, the terminal vertex is determined by the map $\Phi_{G_n}: V(G_n) \to V(G_n)$,

$$\Phi_{G_n}((u(t), l(t)) = \begin{cases} (0, tl(t)), & (u(t) = 0) \\ (tu(t), l(t)), & (u(t) \neq 0) \end{cases}. \tag{25}$$

Theorem 3.2. *The global state transition diagram of LCA-*$(0,1;0;1,1)_n$ *is isomorphic to the graph* G_n.

Proof. Let $f(t)$ be a configuration of LCA-$(0,1;0;1,1)_n$. It is uniquely represented as

$$f(t) = u(t) + l(t), \qquad u(t) \in U_n[t], \quad l(t) \in L_n[t].$$

In the case $u(t) \neq 0$, let

$$u(t) = u_k + \sum_{i=1}^{k-1} x_i u_i, \qquad x_i \in \mathbf{Z}_2. \tag{26}$$

The isomorphism Θ_n from the set of all configurations $P_n[t]$ of LCA-$(0,1;0;1,1)_n$ into $V(G_n)$ is defined as follows. In the case $u(t) \neq 0$,

$$\Theta_n(f(t)) = (u(t), \tau^k(l(t))),$$

where the natural number k is determined by (26) uniquely for each $u(t)$. In the case $u(t) = 0$,

$$\Theta_n(f(t)) = (0, l(t)).$$

Assume that $u(t) \neq 0$. From the definitions of τ_n and Θ_n,

$$\tau_n(f(t)) = u_{k-1} + \sum_{i=1}^{k-2} x_{i+1}u_i + tl(t),$$

$$\Theta_n(\tau_n(f(t))) = \left(u_{k-1} + \sum_{i=1}^{k-2} x_{i+1}u_i,\ \tau^k(l(t))\right).$$

On the other hand, from the definitions of Θ_n and Φ_{G_n},

$$\Theta_n(f(t)) = \left(u_k + \sum_{i=1}^{k-1} x_i u_i,\ \tau^k(l(t))\right),$$
$$\Phi_{G_n}(\Theta_n(f(t))) = \left(u_{k-1} + \sum_{i=1}^{k-2} x_{i+1} u_i,\ \tau^k(l(t))\right).$$

In the case $u(t) = 0$, the following equations hold:

$$\begin{aligned} \tau_n(f(t)) &= tl(t), & \Theta_n(\tau_n(f(t))) &= (0, tl(t)),\\ \Theta_n(f(t)) &= (0, l(t)), & \Phi_{G_n}(\Theta_n(f(t))) &= (0, tl(t)). \end{aligned}$$

Therefore, the following diagram is commutative.

$$\begin{tikzcd} P_n[t] \arrow[r, "\Theta_n"] \arrow[d, "\tau_n"'] & V(G_n) \arrow[d, "\Phi_{G_n}"] \\ P_n[t] \arrow[r, "\Theta_n"'] & V(G_n) \end{tikzcd}$$

Theorem 3.3. *For the eigenpolynomials $\varphi_n(t)$ of A_n, the recurrence formulas hold:*

$$\begin{aligned} &\varphi_n(t) = t\varphi_{n-1}(t) + \varphi_{n-2}(t) + \varphi_{n-3}(t), \qquad (n \geq 4)\\ &\varphi_1(t) = t, \quad \varphi_2(t) = t^2 + 1, \quad \varphi_3(t) = t^3 + 1. \end{aligned}$$

Proof. It is confirmed by expanding the eigenpolynomial $|tI_n - A_n|$.

The eigenpolynomials $\varphi_n(t)$ for $n = 1, \cdots, 10$ are as follows:

$$\begin{aligned} &\varphi_1(t) = t, & &\varphi_2(t) = 1 + t^2,\\ &\varphi_3(t) = 1 + t^3, & &\varphi_4(t) = 1 + t^2 + t^4,\\ &\varphi_5(t) = t + t^2 + t^5, & &\varphi_6(t) = t^4 + t^6,\\ &\varphi_7(t) = 1 + t + t^4 + t^7, & &\varphi_8(t) = t^4 + t^6 + t^8,\\ &\varphi_9(t) = 1 + t + t^5 + t^6 + t^9, & &\varphi_{10}(t) = 1 + t^2 + t^8 + t^{10}. \end{aligned}$$

Note that the coefficients of $\varphi_n(t)$ are elements of $\mathbf{Z}_2$.

Theorem 3.4. *LCA-$(0,1;0;1,1)_n$ is invertible, iff $n \equiv 0, 2, 3, 4\,(mod 7)$.*

Proof. LCA-$(0,1;0;1,1)_n$ is invertible, iff $d_n = 0$. This condition holds, iff $n \equiv 0, 2, 3, 4 \pmod 7$. It is confirmed immediately for $n = 1, \cdots, 7$. In the case $n > 7$, it is confirmed from the recurrence formulas of $\varphi_n(t)$ and by induction on n.

Theorem 3.5. *LCA-$(0,1;0;1,1)_n$ has non-trivial fixed configurations, iff $n \equiv 2, 3\,(mod 4)$.*

Proof. The configuration 0 is the trivial fixed configuration. LCA-$(0,1;\ 0;\ 1,1)_n$ has non-trivial fixed configurations, iff $(t-1)|\varphi_n(t)$. This condition holds, iff $n \equiv 2, 3 \pmod 4$. It is confirmed immediately for $n = 1, \cdots, 4$. In the case $n > 4$, it is confirmed by induction on n.

Theorem 3.6. *The longest transient length of LCA-*$(0,1;0;1,1)_n$ *is* d_n.

Proof. Any configuration of LCA-$(0,1;0;1,1)_n$ is uniquely represented as

$$f(t) = u(t) + l(t), \qquad u(t) \in U_n[t], \quad l(t) \in L_n[t].$$

Assume that $u(t) \neq 0$, and let

$$u(t) = u_k + \sum_{i=1}^{k-1} x_i u_i, \qquad x_i \in \mathbf{Z}_2.$$

The transient length of $f(t)$ is k. The longest transient length of LCA-$(0,1;0;1,1)_n$ is d_n.

Theorem 3.7. *Let* $i \geq 0$ *and* $j \geq 1$ *be the smallest integers that satisfy*

$$t^i \equiv t^{i+j} \qquad (mod\, \varphi_n(t)).$$

Then, $i = d_n$, *and* j *is the longest period length of LCA-*$(0,1;0;1,1)_n$.

Proof. If $k < d_n$, the configuration t^k is represented as

$$t^k = u_k + \tau^{-k}(t^{d_n}).$$

The configuration $\tau^{-k}(t^{d_n})$ is cyclic, but u_k is nilpotent. Therefore, the configuration t^k is not cyclic. On the other hand, the configuration t^{d_n} is cyclic. Therefore, $i = d_n$, and for some integer $j \geq 1$, the equation

$$t^i \equiv t^{i+j} \qquad (\mathrm{mod}\, \varphi_n(t)).$$

holds.

Table 1 shows the longest transient lengths and the longest period lengths of LCA-$(0,1;0;1,1)_n$ in the case $1 \leq n \leq 20$, where n is the cell size, $d(n)$ is the longest transient length (LTL), and $l(n)$ is the longest period length (LPL).

4 Conclusion

Properties of LCA-$(0,1;0;1,1)_n$ are investigated. These properties are determined essentially by the eigenpolynomials of the global state transition matrices.

- The structure of the global state transition diagram is determined.
- The condition for LCA-$(0,1;0;1,1)_n$ to be invertible is determined.
- The condition for LCA-$(0,1;0;1,1)_n$ to have non-trivial fixed configuration is determined.
- The longest transient length is determined.
- The longest period length is determined.

The methods mentioned in this article are useful for other LCA's with some arrangements.

cell size	LTL	LPL
n	$d(n)$	$l(n)$
1	1	2
2	0	2
3	0	3
4	0	6
5	1	15
6	4	2
7	0	62
8	4	6
9	0	186
10	0	16
11	0	1023
12	2	62
13	2	21
14	0	24
15	1	168
16	0	510
17	0	7905
18	0	24
19	1	84
20	2	42

Table 1. The longest transient lengths and the longest period lengths of LCA-$(0, 1; 0; 1, 1)_n$.

References

[1] O. Martin, A. M. Odlyzko, S. Wolfram.: Algebraic properties of cellular automata. Commun. Math. Phys. 93, 219-258 (1984).

[2] P. Guan, Y. He.: Exact results for deterministic cellular automata with additive rules. J. of Statist. Phys. Vol. 43. Nos. 3/4, 463-478 (1986)

[3] E. Jen.: Cylindrical cellular automata. Commun. Math. Phys, 118, 569-590 (1988).

[4] M. Nohmi.: On a polynomial representation of finite linear cellular automata. Bull. of Informatics and cybernetics, 24 No. 3/4, 137-145 (1991)

[5] S. Inokuchi, K. Honda, H, Y. Lee, T. Sato, Y. Mizoguchi and Y. Kawahara.: On reversible cellular automata with finite cell array. Proceedings of Fourth International Conference on Unconventional Computation, LNCS 3699. (2005).

.

How far is to the next recurrent configuration? An NP-complete problem in the sandpile model

Matthias Schulz

University of Karlsruhe, Department for Computer Sciences
Am Fasanengarten 5, 76128 Karlsruhe, Germany
schulz@ira.uka.de

Abstract. We take a look at transient configurations of the Abelian Sandpile Model (ASM). If we add enough grains on the “right” sites to this configuration we get a recurrent configuration of the ASM. In this paper we show that it is NP-complete to decide for a transient configuration c on a grid of size $m \times m$ and a natural number k if it is possible to add k grains of sand to c such that we get a recurrent configuration. We show this by reducing the problem of deciding whether a given planar cubic graph of size $n \in \Theta(m)$ has a vertex cover, which is NP-complete, to this problem.

1 Introduction

The Abelian Sandpile Model (ASM), introduced by Bak, Tang and Wiesenfeld in [1], is the standard example for Self-Organized Criticality (SOC): Grains of sand are added to random sites of a grid, and if one site contains four or more grains of sand, four grains topple from this site to the four von-Neumann neighbors, possibly leading to avalanches.

After some time, we reach a state of Self-Organized Criticality, where the statistic characteristics of the configuration hardly change anymore and are such that the sizes of avalanches triggered by added grains follow a power law. If one considers the ASM a Markov Chain, this state is reached roughly speaking when recurrent configurations are reached. (In fact, the SOC state usually is reached a little before the recurrent configurations are reached.)

Interest for the algebraic properties of the ASM and especially the set of recurrent configurations started with [3]. Among these properties is the fact that the set of recurrent configurations together with a naturally defined addition is an Abelian group.

There has been less interest in the set of transient configurations of the ASM. However, the author found partial orders for the set of transient configurations and measures which tell how “far” a transient configuration is from being recurrent. (cf. [7].)

The most obvious measure would be the minimal number of grains we have to add to a transient configuration to get a recurrent configuration. However, we show in this paper that the equivalent decision problem is NP-complete, which means that the computation of this measure is very hard (if P $\neq$ NP).

2 Basics

Definition 2.1. *For $n \in \mathbb{N}$, we define the basic components of the $m \times m$ Sandpile Model.*

1. *$Z = \{(x, y) \in \mathbb{Z}^2 : 1 \leq x, y \leq n\}$ is the set of sites of the ASM.*
2. *For $(x, y) \in Z$, $N(z) = (z + \{(0, \pm 1)\} \cup \{\pm 1, 0)\}) \cap Z$. (In other words, $N(z)$ is the von-Neumann neighborhood of z in Z without z.)*
3. *A configuration c assigns to each site $z \in Z$ a non-negative number of grains.*
4. *When a site $z \in Z$ fires in a configuration c, the number of grains on z is decreased by four and for each $z' \in N(z)$ the number of grains is increased by one.*
   This means the resulting configuration c' satisfies
   $$c' = c - 4e_z + \sum_{z' \in N(z)} e_{z'},$$
   where e_z is the function which assigns 1 to z and 0 to every other site.
   A site z can only fire if it contains at least four grains of sand.
5. *A configuration c is stable if no site can fire, otherwise unstable. $\mathcal{C}$ denotes the set of stable configurations.*
6. *Starting from an unstable configuration c and letting sites fire until a stable configuration is reached leads to a unique configuration $c_{rel} \in \mathcal{C}$ (cf. [2]). This process is called the relaxation of c and denoted $Rel(c)$. The number of times a site $z \in Z$ fires during $Rel(c)$ is independent of the sequence of the firings.*
7. *The operation $\oplus$ is defined on the set of all configurations (and especially $\mathcal{C}$) by*
   $$c \oplus d = (c + d)_{rel}.$$
   This operation is associative and commutative.
8. *A configuration c is called recurrent if there exists a configuration $i \in \mathcal{C}$ with $i \neq 0$ and $c \oplus i = c$. The set of recurrent configurations is denoted by $\mathcal{R}$. A configuration $c \in \mathcal{C}$ which is not recurrent is called transient.*

In this paper, we will take a closer look at $\mathcal{R}$ and the question how many grains of sand have to be added to a given transient configuration c such that we get a recurrent configuration.

In other words, given the transient configuration c, we are looking for a configuration d containing as few grains as possible such that $c \oplus d \in \mathcal{R}$ holds.

The following lemma helps us redefining this problem:

Lemma 2.1. *Let $b \in \mathcal{C}$ be the configuration with $\forall z \in Z : b(z) = 4 - |N(z)|$. This means that $b(z) = 2$ if z is one of the corners of Z, $b(z) = 1$ if z is on the border of Z and 0 otherwise. Then the following statements hold:*

1. *$c \in \mathcal{R} \iff c \oplus b = c$.*
2. *$c \in \mathcal{R} \iff$ Each site fires once during the relaxation of $c + b$.*
3. *$\forall c \in \mathcal{C}$: No site fires more than once during the relaxation of $c + b$.*
4. *$c \oplus d \in \mathcal{R} \iff$ Each site fires at least once during the relaxation of $c+b+d$.*

The first two claims are proven in [5]. The last two claims are not hard to verify.

Now we can look at the following problem: Given a transient configuration $c \in \mathcal{C}$, how many grains of sand must be contained in a configuration d such that each site fires at least once during $c + d + b$?

This problem is equivalent to finding the least number of grains we have to add to c such that the resulting configuration is recurrent.

In the next section, we will state some facts about the firing of sites when most of the sites contain no more than two grains of sand, which will give us something like an "impact radius" of grains that are added to some regions of Z.

We will then use these facts to construct a configuration c_G for a planar graph G of maximal degree 3 with n vertices such that the following two statements hold:

1. If we add one grain of sand to each "edge" of G to get c', each site fires during $c' + b$.
2. If d is a configuration which contains no grain of sand near an "edge" e of G and each site fires during $c + d + b$, then d contains at least n grains of sand.

We will denote the size of a minimal vertex cover of G by $minVC(G)$.

We make sure that a grain of sand falls onto an "edge" if a grain of sand is added to an adjacent "vertex", such that the first statement guarantees that we need at most $minVC(G)$ grains of sand in d.

The second statement tells us that we have to add a grain of sand near every "edge" in order to add fewer than n grains of sand; from this, we will show how to construct a vertex cover for G that contains at most as many vertices as grains of sand have been added.

3 Dynamic of firings in regions of two

The following statements are not hard to prove, although the proofs are rather inelegant. By "distance" there will always be meant the distance in the maximum norm.

Lemma 3.1. *Let $c \in \mathcal{C}$ be a configuration such that there is a rectangular block $B \subseteq Z$ which satisfies: Each site outside B in a distance of at most k from B contains at most two grains of sand.*

If we add $i \leq k$ grains of sand to B, no site z with distance $\geq i$ from B will fire during the relaxation.

This statement can be proved by a harmless but also charmless induction: If one adds only one grain, the sites at the border of B can fire at most once, and therefore the sites outside B cannot fire. We then repeat this step by substituting B' for B, where B' contains the sites in B as well as the sites with distance 1 to B.

Theorem 3.1. *Let $M \subseteq Z$ be a connected subset of Z, $k, n \in \mathbb{N}$ positive numbers and $c \in \mathcal{C}$ a configuration such that the following statments hold:*

- *All sites z with distance $\leq 2n + 3$ from M contain at most two grains of sand.*
- *There exist rectangular blocks $B_1, \ldots, B_k \subseteq Z$ such that the set of all sites containing three grains of sand outside M is a subset of $\bigcup_{i=1}^{k} B_i$, the distance between two sites in different blocks B_i and B_j is always at least $2n + 3$ and the distance between a site in any block B_i and a site in M is at least $2n+3$.*

So, all sites containing three grains of sand are in rectangular blocks "far away" from M.

If n grains are added to c and a site in M fires during the relaxation, then at least one grain of sand has been added to a site with distance $< 2n + 3$ to M.

This can be proven by a rather uninspiring and ugly induction over k which we skip here. Instead, we refer to intuition and Lemma 3.1:

If one adds a grain of sand to a block B_i, the block containing sites with three grains increases in every direction by at most one, according to Lemma 3.1. This means that no blocks B_i, B_j interact even if we add n grains of sand to each block B_i, so there is no way that these blocks can "help" us if we want to make a site in M fire.

If we add grains of sand outside the blocks and in a distance at least $2n + 3$ from M, the influence of these additions and subsequent firings (again according to Lemma 2) can only go as far as n from any of the sites a grain has been added to.

Therefore, adding n grains of sand to sites in a distance of at least $2n + 3$ from M will leave M unchanged.

This theorem will be very important for our construction later on, since we will have such a configuration as in Theorem 1 when we add grains to "edges" in a way that a given "edge" does not fire. Therefore we have to add a grain near this "edge" or at least n grains elsewhere to make the sites of this "edge" fire.

Using this fact, we can construct a vertex cover whose size is at most the number of grains added to the original configuration.

Theorem 3.2. *Let $V \subseteq Z$ be a Moore-connected subset of Z and B the smallest rectangular block which contains V. Let c be a configuration with $\forall z \in B \setminus V$: $c(z) = 2$.*

If all sites in V fire during $Rel(c)$, all sites in B fire during $Rel(c)$.

Proof. Suppose that this claim is wrong. Let then c be a configuration as specified, where all sites in V fire during $Rel(c)$ and at least one site of B does not fire during $Rel(c)$.

Let P be the set of all sites in B that do not fire during $Rel(c)$ and P' be a maximal von-Neumann connected subset of P. We define $z \in P'$ as the site that lies most to the left, among those most on the top. If the site above z as well as left to z would be in $B \setminus P$, those two sites would fire during $Rel(c)$ and two grains of sand would fall onto z, which afterwards also could fire. Since this would contradict $z \in P$, z has to be on the left border or on the upper border of B.

Without loss of generality, we assume z being on the left border of B. P' then has no connection to the right border of B, since else it would intersect with V. The sites in P' most on the right most on the top and most on the bottom would then have to be on the upper respectively lower border of B. Then P' also would intersect with V, which is a contradiction.

Therefore, the claim must be true.

In our construction we will divide Z into blocks of a certain size; the theorem above will allow us to make sure that the sites of these blocks will fire when we want them to.

4 Preliminaries for the construction

It is shown by Mohar in [6] that the problem of deciding for a natural number $k > 0$ and a planar cubic graph G whether G has a vertex cover of size k is NP-complete.

This clearly also holds for connected planar cubic graphs, which we will use here.

As Kant shows in [4] there exists a linear time algorithm to draw a planar cubic graph with $n > 4$ vertices on an $n \times n$ grid with each edge having at most one bend. Further, at most $\frac{n}{2} + 1$ bends exist in the drawing.

Starting from such a drawing of a connected planar cubic graph G' with k vertices, we first construct a no-bend drawing of a connected planar graph of maximum degree 3 G with $n \in \Theta(k)$ vertices such that it is easy to compute a minimum vertex cover of G' if one has a minimum vertex cover of G.

Consider an edge e that has a bend in the drawing of G'. We add two vertices to e, one of them on the bend. Possibly we have to "stretch" the drawing in one direction such that the other additional vertex is on the grid.

After we do this for every edge containing a bend we have a no-bend drawing of a graph $G = (V, E)$ with n vertices in a grid whose size is at most $n \times n$.

It is easy to verify that $n \in \Theta(k)$ holds. Further, if $2i$ vertices were added during the process, $minVC(G) = minVC(G') + i$. (The proof is fairly simple.) Therefore it follows that if we can find $minVC(G)$ in polynimial time we can determine $minVC(G')$ in polynomial time.

We call (G, i) the *extension* of G'.

Definition 4.1. *Let T be a spanning tree of G, F the set of faces in the no-bend drawing of G.*

1. *$D = (F, E')$ is the graph with $(f_1, f_2) \in E' \iff$ there is an edge $e \in E \setminus T$ which borders on f_1 and f_2. We then denote (f_1, f_2) by $d(e)$. D is a tree.*
2. *The outer face f_r of the drawing shall be the root of D.*
3. *A face $f \in F$ dominates an edge $e \in E \setminus T$ iff f is adjacent to $d(e)$ and the path from f to f_r in D does not contain $d(e)$.*
4. *An edge $e \in E \setminus T$ dominates a face $f \in F$ iff f is adjacent to $d(e)$ and the path from f to f_r contains $d(e)$.*
5. *For $e \in E \setminus T$, let D_e be the graph $D' = (F, E' \setminus \{d(e)\})$. We call the set of edges and faces in the connected component of D_e which does not contain f_r the subtree of e.*

The tree D defines the order of the sites that will fire when manipulating the configuration c_G we will construct:

- During $Rel(c_G + b)$, only the sites in f_r as well as the sites belonging to vertices on the border of f_r will fire.
- After the sites of the face dominating an edge e have fired, all sites on the "edge" in the changed configuration will contain three grains of sand.
- After the sites of the edge dominating the face f have fired, all sites in f and all sites belonging to vertices on the border of f that have not already fired can fire.
- If one adds a grain of sand to every "edge" except one "edge" e (and these are the only grains added), no site belonging to the subtree of e will fire.

5 Construction of c_G

In this section, we construct a configuration $c_G \in \mathcal{C}$ for a planar graph G of maximum degree 3 with n vertices and a given no-bend drawing such that the minimum number of grains that have to be added to c_G in order to get a recurrent configuration (from now on called the *distance* from c_G to $\mathcal{R}$ or $dist(c_G)$) equals $minVC(G)$.

Let Z be a grid of size $((6 + 3n) \cdot (4n + 9)) \times (6 + 3n) \cdot (4n + 9)$, which is divided into blocks of size $(4n + 9) \times (4n + 9)$.

5.1 Elements of construction

There will be four kinds of blocks:

1. Vertex blocks: A vertex block represents a vertex of G. The structure for $n = 2$ of a vertex block is shown in Table1. In the directions of edges incident to e there are "buds" at the end of the lines of sites which contain 3 grains.
2. Edge blocks: An edge block represents a section of an edge $e \in E \setminus T$. All sites in an edge block contain two grains of sand.

3. Tree blocks: A tree block represents a section of an edge $e \in T$. All sites in a tree block contain two grains of sand, except the sites on the horizontal/vertical (depending on the orientation of e) middle line which contain only one grain of sand.
4. Face block: A face block represents a part of a face in the drawing of G. All sites in a face block contain two grains of sand, except the sites on either the horizontal or vertical middle line which contain three grains of sand.

							3	0	3							
							3	3	3							
							3	3	3							
								3								
								3								
								3								
								3								
3	3	3						3								
0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3						3								
								3								
								3								
								3								
								3								
								3								
								3								
								3								

Table 1. A vertex block with edges to the top and to the left. Wherever no number is given, the site contains two grains of sand.

An important feature of the vertex blocks is the fact that all sites except the sites containing no grains of sand (we call these sites the *edge points* of the vertex block) fire once if one of the sites containing three grains of sand fires once. (This can be shown by using Theorem 3.2.) Afterwards the block has the structure in Table 2. When a grain of sand is now added to one of the sites containing three grains of sand, the sites which originally contained no grains of sand can fire and a grain of sand falls onto the neighbor site outside the block — which will belong to the middle of an edge block or to the middle of a tree block.

5.2 Setting vertex, edge and tree blocks

We have a drawing of $G = (V, E)$ without bends in an $n \times n$ grid. If the vertex $v \in V$ has the coordinates (x, y), we put a vertex block at the block with coordinates

0	1	1	1	1	1	1	2	3	2	1	1	1	1	1	1	0
1							3	2	3							1
1							3	3	3							1
1								3								1
1								3								1
1								3								1
1								3								1
2	3	3						3								1
3	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2
2	3	3						3								1
1								3								1
1								3								1
1								3								1
1								3								1
1								3								1
1								3								1
0	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	0

Table 2. The same vertex block after each site except the edge points fired once

$(3+3x, 3+3y)$ and place "buds" in the directions where the edges incident to v go.

If an edge $e \in T$ goes from vertex v to vertex v', all blocks between the corresponding vertex blocks become tree blocks with the line of sites containing only one grain in the same orientation (horizontal or vertical) as e.

All other blocks between two vertices $v, v' \in V$ with $(v, v') \in E$ become edge blocks.

This means that all vertex blocks are on coordinates which are both divisible by three, all edge and tree blocks corresponding to horizontal edges have a second component which is divisible by three and all edge and tree blocks corresponding to vertical edges have a first component which is divisible by three.

5.3 Setting the orientation for face blocks

The blocks at the border of Z become face blocks, the orientation of the middle line always orthogonal to the adjacent border; the orientation does not matter for the blocks in the corners. The rest of the blocks in f_r are set by repeatedly choosing a block B which is neighbor to an already set face block B' whose neighbor block on the other side of B' is not a vertex block, and setting the middle line vertical if this neighbor block is above or below B and horizontal otherwise.

We set the face blocks of other faces f by repeatedly choosing a block B which is neighbor to a block B' which is either an already set face block or an edge block for the edge e which dominates f and whose neighbor block on the other side of B' is not a vertex block and setting it horizontal/vertical if B' is on the left or right/above or below B.

We have to show that such pair of blocks B, B' always exist in a face f as long as not all blocks in f have been set:

Without less of generality, let B' be below B.

If B' is an edge block (and therefore run in the horizontal direction) then the second component of B' must be divisible by 3. This means that the second component of the block above B is $1 \mod 3$ and therefore cannot be a vertex block.

If B' is a face block and the block above B is a vertex block, we can set the blocks B^2 right to B', B^3 above B^2 and therefore right to B and then B with a horizontal middle line without having a conflict with a vertex block. (This follows from the fact that the block above B is a vertex block and all vertex blocks have components divisible by 3.)

The configuration we get after setting all blocks is called c_G.

The *sites of a face f* will be all sites in face blocks of f, all sites except edge points in vertex blocks in the Moore neighborhood of a face block of f and all sites on the side of f from the middle line in edge or tree blocks adjacent to face blocks of f.

Note that a site of a vertex block can be a site of different faces f_1, f_2.

Lemma 5.1. *1. During $Rel(c_G + b)$ all sites of f_r fire once.*

2. Each site in the middle line (of the same orientation as e) of each edge block belonging to an edge $e \in E \setminus T$ contains three grains of sand after the sites of the face f which dominates e have fired once, as do the edge points of the adjacent vertex blocks.

3. Let $e \in E \setminus T$ be an edge, f the face dominating e and f' the face dominated by e. If all sites of f have fired and all sites in the middle lines of edge blocks belonging to e have fired, all sites of f' can fire afterwards.

4. Each site in the middle line (of the same orientation as e) of each tree block for an edge $e \in T$ contains three grains of sand after the sites of the faces f and f' which are adjacent to e have fired, as do the edge points of the adjacent vertex blocks.

Proof. 1. Let $B_1, \ldots, B_k$ be the face blocks of f_r in the sequence in which they were set. We show for $2 \leq i \leq k$ that the sites of B_i can fire during $Rel(c_G + n)$ after the sites of $B_1, \ldots, B_{i-1}$ have fired and that the sites of B_1 can fire during $Rel(c_G + b)$.

B_1 lies at the (without loss of generality upper) border of Z, which means that to each site at the upper border of B_1 a grain of sand is added when b is added to c_G. Since the middle line of sites containing three grains of sand in c_G is orthogonal to the upper border, the site in the middle of the upper border of B_1 contains four grains of sand, the other sites at this border contain three grains of sand and the other grains of the middle line contain three grains of sand.

Therefore all sites on the upper border of B_1 as well as all sites on the middle line can fire, and according to Theorem 2 all sites in B_1 can fire.

Consider B_i with $i \geq 2$. Then either B_i lies at the border of Z, in which case the argumentation is analogous to B_1, or the middle line is orthogonal to the border next to a block B_j with $j < i$, according to the construction. In the latter case, each site on this border has received a grain of sand from B_j as the sites of B_j have toppled. As for B_1 this means that all sites in B_i can fire during the relaxation.

Now, consider the polygon made by the outer vertex, edge and tree blocks. This polygon P has at least one convex vertex; in this vertex there has to be a vertex block L_1, since the drawing of G is without bends.

We will start with L_1 and go counter-clockwise along the polygon to the blocks $L_2, \ldots, L_j$. In case of L_i being a vertex block, we will show that all sites in L_i except the edge points fire; if L_i is an edge or tree block, we show that all sites of L_i on the side of f_r fire.

L_1 is a vertex block, therefore it contains a site with three grains of sand at each border. One of these borders lies towards a face block of f_r, since L_1 is on a convex vertex of P. Therefore, this site fires and afterwards all other sites in L_1 except the edge points fire, as described above.

Now, we consider L_i for $i \geq 2$ and assume that all sites of L_{i-1} on the side of f_r have fired.

If L_i is a vertex block there is an edge point on the side to L_{i-1} and a grain of sand has fallen onto the site containing three grains of sand on the side of f_r next to this edge point. Therefore, all sites containing three grains of sand in L_i can fire and therefore all sites in L_i except the edge points can fire.

If L_i is an edge or tree block there is a face block of f_r in a direction orthogonal to the direction where L_{i-1} lies. Since the sites of this face block and the sites of L_{i-1} on the side of f_r have fired, the sites on the border to L_{i-1} on the side to f_r contain three grains of sand, the sites on the border to the face block contain three grains of sand and the site at the intersection of these borders contains four grains of sand.

Using Theorem 3.2, we find that all sites of L_i on the side of f_r fire.

This proves the claim.

2. After all sites of f have have fired once, it is clear from the construction that the sites of the middle line of edge blocks belonging to e contain three grains of sand. Also, if no further grains of sand are added, none of these sites gets another grain of sand, so these sites do not fire.
3. Let $B_1, \ldots, B_k$ be the sequence of blocks from one vertex block adjacent to e to the other vertex block adjacent to e along the edge blocks of e. Without loss of generality these blocks go from left to right.

   After the sites belonging to middle lines of edge blocks belonging to e (or shorter, sites of e) have fired, the edge points of the adjacent vertex blocks also can fire.

   Since all sites of B_1 except the edge points have fired, B_2 has three grains of sand on each site on the left border except the middle site.

After the sites in the middle of B_2 have fired, the sites on the left border of B_2 can fire. This means with Theorem 2 that all sites of B_2 on the side of f' can fire.

Let $i \geq 3$. After B_{i-1} has fired, a grain has been added to each site on the left border of B_i. Again, all sites in B_i on the side of f' can fire.

Therefore all sites in the blocks $B_2, \dots B_{k-1}$ on the side of f' can fire after the sites of f and e have fired.

Let $F_1, \dots F_l$ be the sequence of face blocks of f' as they were set in the construction.

Since F_1 borders on an edge block of e (without loss of generality, this block is below F_1), a grain will be added to each site of the lower border of F_1. According to construction, F_1 has a vertical middle line of sites containing three grains of sand; these sites can now fire, as well as the sites on the lower border of F_1. This means with Theorem 3.2 that all sites of F_1 can topple.

Let $i \geq 2$. If F_1 is a w.l.o.g. upper neighbor of an edge block belonging to e and has a vertical middle line of sites containing three grains, the argumentation that all sites in F_i fire is as for F_1.

Else, F_i is a w.l.o.g. right neighbor of a face block F_j with $j < i$ and a horizontal middle line. Since all sites in F_j fire according to our induction hypothesis, a grain of sand is added to each site on the left border of F_i. The argumentation that all sites of F_i fire is now as for F_1.

Let $L_1 = B_1, L_2, \dots, L_j = B_k$ be the sequence of edge, tree and vertex blocks around f' that does not go over the edge blocks of e. Via induction as in the first proof in this lemma, it can be shown that all sites of all L_i that lie on the side of f' fire.

This proves the claim.

4. Originally, the middle lines of tree blocks contain one grain of sand. After the sites of the adjacent faces have fired, these sites contain three grains of sand, as do the edge points of the adjacent vertex blocks.

Theorem 5.1. *Let $V' \subseteq V$ be a minimum vertex cover of G.*

If one grain is added to the center cell of each vertex block corresponding to a vertex $v \in V'$ to get c', all sites can fire during $Rel(c' + b)$.

Proof. First, we let all center sites of vertex blocks corresponding to vertices in V' fire once. Then there are sites with four grains of sand on each of the four "branches" of these vertex blocks.

Since all sites of f_r fire once during $Rel(c_G + b)$, they will do so for $Rel(c' + b)$.

Let $e = (u, v) \in E \setminus T$ be an edge dominated by f_r and u a vertex in V' which is incident to e.

There is a site with four grains on the branch of the vertex block belonging to u. If we let the sites of this branch fire, a grain of sand will be added to the edge point to e, and all sites of e can fire, since they contain three grains of sand after the sites of f_r have fired once.

Afterwards the sites of the face dominated by e can fire.

We repeat these steps (choosing an edge e whose sites have not fired yet but is dominated by a face whose sites have fired, letting the branch of the adjacent vertex block a grain has been added to fire, letting the sites of e and the face dominated by e fire) until all sites in all face blocks, edge blocks and vertex blocks have fired (the edge points to an edge e can fire when the sites of e fire).

Afterwards, all sites in the middle lines of tree blocks contain three grains of sand and can be set firing by letting the branches of the adjacent vertex blocks a grain has been added fire.

Then all sites in Z have fired.

Corollary 5.1. *$dist(c_G) \leq minVC(G)$.*

Proof. In Theorem 5.1, we added $minVC(G)$ grains to c_G in order to get a recurrent configuration; the minimal number of grains therefore cannot be greater.

Lemma 5.2. *1. Let $e \in E \setminus T$ be an edge.*
If a grain of sand is added to the center site of each edge block belonging to an edge $e' \neq e$ which is not in the subtree of e and to each center site of each tree block and afterwards b is added, we get a configuration c' which satisfies the condition for Theorem 3.1 with M being the set of sites in the middle lines of the edge blocks of e, the sites of the adjacent vertex blocks and the sites adjacent to these vertex blocks.

2. Let $e \in T$ be an edge in the spanning tree of G.
If a grain of sand is added to the center site of each edge block and each tree block not belonging to e and afterwards b is added, we get a configuration c' which satisfies the condition for Theorem 3.1 with M being the set of sites in the middle lines of the tree blocks of e, the sites of the adjacent vertex blocks and the sites adjacent to these vertex blocks.

Proof. 1. Since there are no firings after the grains have been added to the edge and tree blocks to get the configuration $c \in \mathcal{C}$, each site fires at most once during $Rel(c + b)$.

Let R be the set of all sites z that satisfy one of the following conditions:

- z lies in an edge block belonging to e and lies on the middle line or on the side of the subtree of e.
- z lies in a face block belonging to a face in the subtree of e.
- z lies in an edge block belonging to an edge in the subtree of e.
- z lies in a tree block adjacent to the subtree of e either on the middle line or on the side of the subtree of e.
- z lies in a vertex block which is adjacent only to faces in the subtree of e.

The border of R consists of the sites of e, the sites of edges $e' \in T$ and sites adjacent to vertex blocks; we will call this border R'. Note that all convex vertices of R' are at the end of edges.

Suppose that some of the sites in R' fire during $Rel(c + b)$ and let z be the first site to fire.

Each site of R' initially contains at most two grains of sand, which means that two grains of sand must have fallen onto z before it fired.

Since R' is a closed polygon, no neighbor of z could have fired before z fired and no site inside R can fire before a site of R' has fired, z must be a convex vertex of R'.

This means that z has two neighbors in R' and an edge point as third neighbor. This edge point z' cannot have fired before z, since its other three neighbors could only have given z' three grains of sand, which are not enough to let z' fire.

Therefore, at most one grain of sand can have been added to z' before it fired, which is a contradiction.

Therefore no site in R' (and therefore R) fires during $Rel(c+b)$.

On the other hand it can be shown that all sites outside R fire during $Rel(c+b)$, similarly to the proof of Theorem 5.1.

This means that all sites of e and all sites in R on the border to vertex blocks adjacent to R contain three grains of sand. Further, the only sites containing more grains in $c \oplus b$ than in c are the sites in R': No grain fell onto a site in $R \setminus R'$ and each site in $Z \setminus R$ lost four grains during firing at most gaining four grains from firing neighbors or the addition of b.

Therefore all sites with three grains of sand outside M as given above are

- center sites of edge blocks or tree blocks on the border of R (blocks of size 1×1 which have a distance of at least $2n+3$ to any site in another face, edge, tree or vertex block or on the border of the same edge block);
- middle lines of face blocks (blocks of size $1 \times (4n+9)$ which have a distance of at least $2n+3$ to any site in another face/edge/tree/vertex block or on a border parallel to the middle line);
- sites in or adjacent to a vertex block (blocks of size $(4n+11) \times (4n+11)$ which have a distance of at least $2n{+}3$ to any site in a face/edge/tree/vertex block not adjacent to the vertex block and to any site in the middle line of a face block).

We call the set containing the sites of a vertex block B and the sites adjacent to B the *extended vertex block* of B.

Because

- no extended vertex block not in M is nearer to M, a center site of an edge or tree block, the middle line of a face block or another vertex block than $2n+3$,
- no adjacent set of middle lines of face blocks is nearer to M, a center site of an edge or tree block, the middle line of another face block or a vertex block than $2n+3$,
- no center site of an edge or tree block is nearer to M, a center site of another edge or tree block, the middle line of a face block or a vertex block than $2n+3$,

the claim is proven.

2. Let c be again the configuration we get when adding the grains to each edge and tree block not belonging to e. As in the proof to Theorem 5.1, all sites not belonging to the middle lines of the tree blocks of e or being edge points to e fire during the relaxation.

On the other hand the line from one edge point to the next via e is a straight line beginning and ending with a site containing no grains of sand and having only sites containing one grain of sand in between. It is easy to show that no site in such a line can fire during $Rel(c+b)$.
Since all other sites than the ones in this line fire, each site in such a line contains three grains of sand in c'.
Analogously to the proof above, it is easy to see that all sites containing three grains of sand are in blocks which have at least the distance $2n+3$ to one another and to M. (The blocks are of the same categories as above.)
Therefore the claim is proven.

Corollary 5.2. *Let $e \in E$ be an edge. If for a configuration d the configuration $c_G \oplus d$ is recurrent and d contains at most n grains of sand, d contains a grain of sand at a site with a distance at most $2n+3$ from the middle line of an edge/tree block belonging to e or from a vertex block belonging to a vertex incident to e.*

Proof. If $c_G \oplus d \in \mathcal{R}$ holds we know that each site fires at least once during $Rel(c_G + d + b)$.

This also means that all sites that did not fire during the relaxation in Lemma 5.2 will fire if one further adds d to the resulting configuration c', since if all sites fire during $Rel(c_G + d + b)$, they will fire during $Rel(c_G + d' + b + d)$, where d' is the configuration with one grain at the center site of the edge blocks specified in Lemma 4.

We have shown that the sites of e do not fire during the relaxation in Lemma 5.2.

Theorem 3.1 now tells us that these sites only can fire if one of the at most n grains added to the configuration has to be added "near" the middle line of e or near one of the vertex blocks adjacent to e.

Corollary 5.3. *$dist(c_G) \geq minVC(G)$.*

Proof. Let d be a configuration containing a minimal number of grains $|d|$ such that $c_G \oplus d \in \mathcal{R}$.

We know that $minVC(G)$ grains are sufficient and therefore know that $|d| < n$. This means that there is a grain "near" each "edge" $e \in E$.

A site can only be "near" (i.e. having at most the distance $2n+3$ from) two edges if it is "near" a vertex incident to both edges.

We number the set $E = \{e_1, \ldots, e_{|E|}\}$ and construct the set $V_{|E|}$, starting with $V_0 = \emptyset$ and defining $V_{i+1} = V_i \cup \{v_{i+1}\}$, where v_{i+1} is a vertex incident to $e_{i+1} = (u, v)$, chosen by the following rules:

If there is a grain in $|d|$ "near" u, $v_{i+1} = u$. Else, there must still be a grain "near" e_{i+1} and $v_{i+1} = v$.

It is easy to see that $V_{|E|}$ is a vertex cover of G, since for every edge (u, v) u or v is in $V_{|E|}$.

Further, we get at most one vertex for every grain of sand in d near one of the edges or vertices: During the construction of $V_{|E|}$ we can assign to each grain "near" an edge e_i the vertex v_i without getting conflicts, since a double assignment can only take place if a grain was “near” two edges, in which case it was “near” a common incident vertex v which gets assigned to the grain both times.

Therefore $dist(c_G) = |d| \geq minVC(G)$.

Theorem 5.2. *The problem to decide for a configuration $c \in \mathcal{C}$ and a natural number k whether $dist(c) \leq k$ is NP-complete.*

Proof. If one had a deterministic algorithm with polynomial time complexity to solve this decision problem, one could construct the configuration c_G and the number $k+i$ for the extension (G, i) of a cubic planar graph G' as described above and decide in a polynomial time complexity whether $k+i$ grains are sufficient to add to c_G in order to get a recurrent configuration. This is the case iff $minVC(G) \leq k+i$ which is the case iff $minVC(G){\leq}k$.

Since the problem to decide this last question is NP-hard, the decision problem for *dist* is NP-hard.

Also, the decision problem whether k grains of sand are enough to get a recurrent configuration by adding these grains to a configuration $c \in \mathcal{C}$ lies in NP since one can check with polynomial time complexity whether a given configuration d with k grains satisfies $(c+d)_{rel} \in \mathcal{R}$. (cf. [2])

Therefore, the decision problem we proposed is NP-complete.

Corollary 5.4. *Let c be a configuration not necessarily in $\mathcal{C}$ and $k \in \mathbb{N}$ a natural number.*

It is NP-hard to decide whether a configuration d with k grains exists such that each site fires at l^east once during $Rel(c+d)$.

Proof. Let $c' \in \mathcal{C}$ be a transient configuration and $c = c' + b$. Then the decision problem for c is equivalent to the problem of deciding whether a configuration d with k grains exists such that $c' \oplus d \in \mathcal{R}$.

Therefore the problem is NP-hard.

6 Conclusion

We have shown for the two-dimensional ASM that it is NP-complete to decide whether k grains of sand can be added to a transient configuration in order to get a recurrent configuration.

If we look at n-dimensional Sandpile Models (where sites can fire when they contain at least 2^n grains of sand and lose them to their von-Neumann neighbors), we can use an analogous construction to show that the problem analyzed in this paper is NP-complete for these sandpiles, too. (In fact, an easier construction for the three-dimensional Sandpile Model has been introduced in [8].) Further, it is quite easy to see that this problem lies in P for the one-dimensional Sandpile Model. This means that we know for all n whether the problem is NP-complete.

Further research could look at this problem for other families of graphs (e.g. trees, for which this problem also lies in P).

References

[1] Per Bak, Chao Tang, and Kurt Wiesenfeld, *Self-organized criticality: An explanation of the 1/f noise*, Phys. Rev. Lett. **59** (1987), 381–384.

[2] F. Chung and R. Ellis, *A chip-firing game and Dirichlet eigenvalues*, Discrete Mathematics **257** (2002), 341–355.

[3] Deepak Dhar, *Self-organized critical state of sandpile automaton models*, Phys. Rev. Lett. **64** (1990), 1613–1616.

[4] Goos Kant, *Drawing planar graphs using the canonical ordering*, Algorithmica **16** (1996), no. 1, 4–32.

[5] S. N. Majumdar and Deepak Dhar, *Equivalence between the Abelian sandpile model and the $q \to 0$ limit of the Potts model*, Physica A: Statistical and Theoretical Physics **185** (1992), 129–145.

[6] Bojan Mohar, *Face covers and the genus problem for apex graphs*, J. Comb. Theory, Ser. B **82** (2001), no. 1, 102–117.

[7] Matthias Schulz, *Measures for transient configurations of the sandpile-model*, ACRI, 2006, pp. 238–247.

[8] ———, *An NP-complete problem for the Abelian Sandpile Model*, Complex Systems **17** (2007), 17–28.

Changing the neighborhood of CA: local structure, equivalence and reversibility

Hidenosuke Nishio*[1] and Thomas Worsch[2]

[1] ex. Kyoto University,
Iwakura Miyake-cho 204-1, Sakyo-ku, 606-0022 Kyoto
YRA05762@nifty.com
[2] Faculty of Informatics, University of Karlsruhe,
am Fasanengarten 5, D-76128 Karlsruhe
worsch@ira.uka.de

Abstract. From the definition of a cellular automaton (S, Q, f, ν) with S a discrete cellular space, Q a finite set of cell states, f an n-ary local function $f(x_1, ..., x_n)$ and ν a neighborhood function $\nu : \{1, ..., n\} \to S$, we pick up a pair (f, ν) called the *local structure*. By defining the local structure, new aspects of changing the neighborhood can be seen. For instance, the following lemma is given; If (f, ν) and (f', ν') are two reduced local structures which are equivalent, then there is a permutation π such that $\nu^\pi = \nu'$. As a corollary the previous theorem in the MCU paper is proved; By changing the neighborhood, infinitely many different CA are induced from any single local function. The general notions of equivalence and isomorphism of CA are reexamined and classification of CA with respect to reversibility, injectivity and surjectivity is discussed in the context of changing the neighborhood. This paper is a successor of Nishio's MCU2007 paper.

1 Introduction

Most studies on cellular automata (CA for short) first assume some standard neighborhood (von Neumann, Moore) or its modifications and then investigate the global behaviors and mathematical properties or look for a local function that would meet a given problem. In 2003, however, H. Nishio and M. Margenstern began a general study of the neighborhood in its own right, where the neighborhood N can be an arbitrary finite subset of the space S and particularly discussed the problem if N generates (*fills*) S or not [7]. On the other hand, as for the dynamics of CA, it has been shown that some properties depend on the choice of the neighborhood, while others do not [5]. In particular, reversibility (injectivity and surjectivity) were examined by means of Java program and simulator, which were coded for the case of arbitrary neighborhoods [9][6]. On

* corresponding author

the other hand, T. Worsch and H. Nishio (2007) treated a new construction for achieving universality of CA by changing the neighborhood [12, 11].

This paper is a successor of Nishio's MCU2008 paper [6] and newly introduces a notion of the *local structure* and reveals new aspects of changing the neighborhood of CA.

The paper is structured as follows; Section 2 gives the definitions of *the local structure* (f, ν) and related terms. In Sect. 3 we give some basic results including a lemma: *If* (f, ν) *and* (f', ν') *are two reduced local structures which are equivalent, then there is a permutation* π *such that* $\nu^\pi = \nu'$. A corollary to this lemma gives another simple proof for the first theorem shown in [6]: *By changing the neighborhood function, infinitely many different CA functions are induced from any single local function.* The results are generally given without proofs, which will be found, say, in Nishio and Worsch (2008) [8]. Section 4 treats another notion of *isomorphism* of local structures and it is shown that the similar corollaries hold. In Sect. 5 classification of CA (local structures) is discussed from the stand point of changing the neighborhood.

2 Definitions

2.1 Cellular automaton CA (S, Q, f, ν)

A cellular automaton (CA for short) is defined by a 4-tuple (S, Q, f, ν):

- S: a discrete cellular space such as $\mathbb{Z}^d$, hyperbolic space ...
- Q: a finite set of the states of each cell.
- $f : Q^n \to Q$: a local function in $n \geq 1$ variables.
- ν: an injective map from $\{1, ..., n\}$ to S, called a *neighborhood function*, which connects the i-th variable of f to $\nu(i)$. That is, $(\nu(1), ..., \nu(n))$ becomes a neighborhood of size n.

In order to study effects of changing the neighborhood (function), we pick up the pair (f, ν) called a *local structure* of CA and investigate its mathematical properties. For the sake of easy understanding, in the following we assume that $S = \mathbb{Z}^d,\ d \geq 1$.

2.2 Local structure (f, ν)

Definition 2.1. *[neighborhood]*
For $n \in \mathbb{N}$*, a neighborhood (function) is a mapping* $\nu : \mathbb{N}_n \to \mathbb{Z}^d$*, where* $\mathbb{N}_n = \{1, 2, \ldots, n\}$*. This can equivalently be seen as a list* ν *with* n *components;* $(\nu_1, \ldots, \nu_n)$*, where* $\nu_i = \nu(i), 1 \leq i \leq n$.

The set of all neighborhoods of *size* n will be denoted as $\mathcal{N}_n$.

Definition 2.2. *[local structure, reduced]*
A pair (f, ν) of a local function $f : Q^n \to Q$ and a neighborhood $\nu \in \mathcal{N}_n$ is called a local structure. We will sometimes use τ for representing a local structure. We call n the arity of the local structure.

A local structure is called reduced, if and only if the following conditions are fulfilled:

- *f depends on all arguments.*
- *ν is injective, i.e. $\nu_i \neq \nu_j$ for $i \neq j$ in the list of neighborhood ν.*

Each local structure induces the *global function* $F : Q^{\mathbb{Z}^d} \to Q^{\mathbb{Z}^d}$ or the dynamics of CA. Every element $c \in Q^{\mathbb{Z}^d}$ is called a *(global) configuration.* For any global configuration $c \in Q^{\mathbb{Z}^d}$ and $x \in \mathbb{Z}^d$, let $c(x)$ be the state of cell x in c. Then F is given by

$$F(c)(x) = f(c(x + \nu_1), c(x + \nu_2), ..., c(x + \nu_n)).$$

2.3 Equivalence

Definition 2.3. *[equivalence]*
Two local structures (f, ν) and (f', ν') are called equivalent, if and only if they induce the same global function. In that case we sometimes write $(f, \nu) \approx (f', \nu')$.

Lemma 2.1.
For each local structure (f, ν) there is an equivalent reduced local structure (f', ν').

Note that for a local structure, the equivalent reduced local structure is *not unique.* As a simple example consider the local function $f(x_1, x_2)$ over $GF(2) : (x_1, x_2) \mapsto x_1 + x_2$ (mod.2). Since the order of the arguments x_i does not matter for the value $f(x_1, x_2)$, the local structures $(f, (0, 1))$ and $(f, (1, 0))$ are equivalent. At the same time both are obviously reduced.

2.4 Permutation of local structure

Definition 2.4. *[permutation of local structure]*
Let π denote a permutation of the numbers in $\mathbb{N}_n$.

- *For a neighborhood ν, denote by ν^π the neighborhood defined by $\nu^\pi_{\pi(i)} = \nu_i$.*
- *For an n-tuple $\ell \in Q^n$, denote by ℓ^π the permutation of ℓ such that $\ell^\pi(i) = \ell(\pi(i))$ for $1 \leq i \leq n$.*

  For a local function $f : Q^n \to Q$, denote by f^π the local function $f^\pi : Q^n \to Q$ such that $f^\pi(\ell) = f(\ell^\pi)$ for all ℓ.

In the first part of the definition we have preferred the given specification to the equally possible $\nu^\pi_i = \nu_{\pi(i)}$, because the former leads to a slightly simpler formulation of the following lemma.

3 Results

Lemma 3.1.
(f,ν) and (f^π,ν^π) are equivalent for any permutation π.

We are now going to show that for *reduced* local structures, the relationship *via a permutation* is the *only* possibility to get equivalence.

Lemma 3.2.
If (f,ν) and (f',ν') are two reduced local structures which are equivalent, then there is a permutation π such that $\nu^\pi = \nu'$.

Lemma 3.3.
If (f,ν) and (f',ν') are two reduced local structures which are equivalent, then there is a permutation π such that $(f^\pi,\nu^\pi) = (f',\nu')$.

By choosing different neighborhoods which are not permutations of each other, one immediately gets the following corollary, which claims the same thing as Theorem 1 of H.Nishio, MCU2007 [6]: *By changing the neighborhood function ν, infinitely many different global CA functions are induced by any single local function $f_3(x,y,z)$ which is not constant.* Proof was given for 1-dimensional CA by concretely showing biinfinite words which correspond to different neighborhoods.

Corollary 3.1.
For each reduced non-constant local function f, there are infinitely many reduced neighborhoods ν, such that the local structures (f,ν) induce pairwise different global CA functions.

4 Isomorphism

Since the above definition of equivalence is *too strong*, we will consider a weaker notion *isomorphism* which allows permutation of the set of cell states. In this respect, we should notice some historical definitions of isomorphism and homomorphism of CA [4][2], though their research topics is different from ours.

In the same space S, consider two CA A and B having different local structures (f_A,ν_A) and (f_B,ν_B), where f_A and f_B are defined on possibly different domains; $f : Q_A^n \to Q_A$ and $f_B : Q_B^{n'} \to Q_B$.

Definition 4.1.
If $|Q_A| = |Q_B|$, then we can consider a bijection $\varphi : Q_A \to Q_B$. Two CA A and B are called isomorphic under φ denoted by $A \underset{\varphi}{\sim} B$, if and only if the following diagram commutes for all global configurations. Note that bijection φ naturally extends to $\varphi : Q_A^{\mathbb{Z}^d} \to Q_B^{\mathbb{Z}^d}$.

$$\begin{tikzcd}
c_A \arrow[r, "\varphi"] \arrow[d, "F_A"'] & c_B \arrow[d, "F_B"] \\
c'_A \arrow[r, "\varphi"] & c'_B
\end{tikzcd}$$

where c_A (c_B) is a global configuration of A (B) and c'_A (c'_B) is the next configuration of $c_A(c_B)$.

Both equivalence and isomorphism of local structures are evidently equivalence relations.

From the definitions of equivalence and isomorphism among local structures, we have

Lemma 4.1.
If $(f_A, \nu_A) \approx (f_B, \nu_B)$, then $(f_A, \nu_A) \underset{\varphi}{\sim} (f_B, \nu_B)$ for any φ. The converse is not always true.

For the isomorphism too, the following lemma is proved in the same manner as Lemma 3.2.

Lemma 4.2 (Lemma 3.2').
If (f_A, ν_A) and (f_B, ν_B) are two reduced local structures which are φ-isomorphic under a bijection $\varphi : Q_A \to Q_B$, then there is a permutation π such that $\nu_A^\pi = \nu_B$.

5 Classification of CA

Classification is a typical problem in the CA study and there are several stand points of classification. For example, CA are classified by the *complexity of dynamical behavior* — fixed points, limit cycles, chaotic and so on, see Chapter 8 of [2] for old references. Note that those past papers assume a standard neighborhood like von Neumann neighborhood and classify the set of local functions. We will investigate, however, the classification problem from a different point of view — by changing the neighborhood. As discussed above, without loss of generality we shall restrict ourselves to the *reduced* local structures. The decision problem of equivalence and isomorphism of local structures is evidently decidable.

Fix a state set Q such as $|Q| \geq 2$ and think of all local functions $f : Q^n \to Q$ of arity n and arbitrary neighborhoods ν of size n in $\mathbb{Z}^d,\ d \geq 1$.

5.1 Linear CA

CA map $F : C \to C$ is linear if and only if $F(\alpha c + \beta c') = \alpha F(c) + \beta F(c')$, where $c, c' \in C$ and $\alpha, \beta \in Q$. A local function f is linear if $f = \sum_{i=1}^{n} a_i x_i,\ a_i \in Q,\ 1 \leq i \leq n$.

Lemma 5.1.
Let F_τ be the global map induced by a local structure $\tau = (f, \nu)$. Then F_τ is linear if and only if f is linear while ν is arbitrary.

Proof: If part is obvious. Only if part is shown by counter example. Let $Q = GF(2)$ and local function of arity 2: $f(x_1, x_2) = x_1 \cdot x_2$. Assume $c(0) = 0, c(1) = 1$ and $c'(0) = 1, c'(1) = 0$. Then $F(c)(0) = F(c')(0) = 0$ i.e. $F(c)(0) + F(c')(0) = 0$, while $F(c + c')(0) = 1$. ■

5.2 Reversible, injective and surjective CA

First we consider reversible 6 Elementary CA, see page 436 of [10].

	000	001	010	011	100	101	110	111
R15	1	1	1	1	0	0	0	0
R51	1	1	0	0	1	1	0	0
R85	1	0	1	0	1	0	1	0
R170	0	1	0	1	0	1	0	1
R204	0	0	1	1	0	0	1	1
R240	0	0	0	0	1	1	1	1

Rules 15, 51 and 85 are equivalent (and isomorphic) each other. Rules 170, 204 and 240 are equivalent (and isomorphic). However, rules 15 and 240 (resp. 51 and 204, 85 and 170) are *not equivalent but isomorphic* under $\varphi : 0 \mapsto 1, 1 \mapsto 0$. Summing up, all reversible ECA are isomorphic.

Remarks The paper by L. Chua et. al. [1] as well as one by Guan et.al. [3] focuses on certain geometrical symmetries of the unit cubes corresponding to local functions of ECA and defines the *global equivalence* which classifies 256 ECA into 88 classes. For instance, it classifies rule 15 and 85 as *globally equivalent* but not 51. This is because our isomorphism considers *all* permutations of neighborhoods, while their global equivalence does not.

If $|Q| = 3$, reversibility is not preserved from changing the neighborhood. See Proposition 9. in [6].

Characterization or classification of local structures which induce not injective and surjective CA is an interesting problem, but we have not yet obtained definite results. As a computational example, we can tell something about rule 30 and 6 permutations of $\pi_0 = (-1, 0, 1)$, see below. By using a Java program *catest*, we see that 6 local structures $\{(30, \pi_i),\ i = 0, .., 5\}$ are all not injective, while $(30, \pi_0)$ and other 3 ones are surjective but $(30, \pi_1)$ and $(30, \pi_3)$ are not.

6 permutations of $(-1, 0, 1)$

$$\pi_0 = (-1, 0, 1), \pi_1 = (-1, 1, 0), \pi_2 = (0, -1, 1),$$
$$\pi_3 = (0, 1, -1), \pi_4 = (1, -1, 0), \pi_5 = (1, 0, -1).$$

6 Concluding remarks

In this paper we defined the local structure of CA and characterized equivalence of CA principally based on the permutation π of local structures. Classification of local functions was discussed from the point of view of changing the neighborhood. A brief comparison with the past works made by some authors was made. There are several problems to be solved: look for an efficient algorithm for deciding if two reduced local structures are equivalent or isomorphic, are there irreversible ECA which become reversible by permuting or changing the neighborhood? and many more.

References

[1] Chua, L., Sbitnev, V., Yoon, S.: A nonlinear dynamics perspective of Wolfram's New Kind of Science. Part III: Predicting the unpredictable, *Int. Journal of Bifurcation and Chaos*, **14**, 2004, 3689–3820.

[2] Garzon, M.: *Models of Massive Parallelism, Analysis of Cellular Automata and Neural Networks*, Springer, 1995.

[3] Guan, J., Shen, S., Tang, C., Chen, F.: Extending Chua's global equivalence theorem on Wolfram's New Kind of Science, *Int. Journal of Bifurcation and Chaos*, **17**, 2007, 4245 – 4259.

[4] H.Yamada, S.Amoroso: Structural and behavioral equivalencies of tessellation automata, *Information and Control*, **18**, 1971, 1–31.

[5] Nishio, H.: How does the neighborhood affect the global behavior of cellular automata?, *Proceedings of ACRI2006, eds. El Yacoubi, B. Chopard and S. Bandini*, LNCS 4173, 2006.

[6] Nishio, H.: Changing the neighborhood of cellular automata, *Proceedings of MCU2007, eds. J. Durand-Lose and M. Margenstern*, LNCS 4664, 2007.

[7] Nishio, H., Margenstern, M., von Haeseler, F.: On algebraic structure of neighborhoods of cellular automata –horse power problem–, *Fundamenta Informaticae*, **78(3)**, 2007, 397–416.

[8] Nishio, H., Worsch, T.: Local structure of cellular automata, Proceedings (kôkyûroku) of RIMS workshop held in January, 2008, to appear.

[9] Nishio, H., Worsch, T.: Neighborhood function for CA, RIMS kôkyûroku vol. 1554, May 2007.

[10] Wolfram, S.: *A New Kind of Science*, Wolfram Media, Inc., 2002.

[11] Worsch, T., Nishio, H.: Achieving universality of CA by changing the neighborhood, accepted for publication, J. Cellular Automata, Proceedings of AUTOMATA 2007 (13th Workshop on CA) held in Toronto, August 2007.

[12] Worsch, T., Nishio, H.: Variations on neighborhoods in CA, *Proceedings of Eurocast 2007, eds. R. Moreno-Diaz et. al.*, LNCS 4739, 2007.

On Brownian cellular automata

Jia Lee[1] and Ferdinand Peper[2,3]

[1] Celartem Technology Inc., Tokyo, Japan
[2] National Institute of Information and Communications Technology, Nano ICT Group, Kobe, Japan
[3] University of Hyogo, Division of Computer Engineering, Himeji, Japan

Abstract. Brownian Cellular Automata are asynchronously timed models that allow certain configurations—like signals—to fluctuate in the cell space over time in random semi-controlled ways. We present such a model and prove its computational universality by showing primitive configurations on the cell space from which computational structures can be constructed. The proposed model has 3-state cells, which are governed by three transition rules. Key to the model's operation is the exploitation of random fluctuations to search for solutions in computational state space. We show how to do this, and how to speed up the searching process such that it becomes competitive with traditional computation methods on CA. Future designs of computers based on devices with nanometer-scale feature sizes may require fluctuating behavior of signals, like in the proposed model, as an important ingredient to achieve efficient operation.

1 Introduction

Cellular Automaton (CA) models are increasingly attracting interest as architectures for computers based on nanometer-scale electronic devices. Having a regular structure, they offer much potential for fabrication by molecular self-assembly techniques [19]. Even in such architectures, though, many issues remain to be addressed, such as reducing power consumption and improving fault-tolerance. The first issue, reducing power consumption, has attracted various proposals, from using reversible computation in CA (e.g. [14]) to timing the updates of cells asynchronously [19]. The second issue, improving fault-tolerance, has focused on limiting the effects of noise and fluctuations (e.g. [9]).

Already considered a problem in the current pre-nanocomputer era, noise and fluctuations are expected to become a major factor interfering with the operation of nanometer-scale electronic devices, to the extent that they cannot be coped with by traditional techniques, which are limited to the suppression of noise and fluctuations and the correction of errors caused by them. This raises the question whether there are more powerful techniques that can exploit noise and fluctuations as an important ingredient in the operation of nanocomputers.

Operation at room temperatures is the prevailing state of affairs in VLSI technology, and, to be competitive, nanoelectronics will have to conform to that.

The noise and fluctuations accompanying room temperatures, however, will affect the fidelity of operations taking place at nanometer scales—reason to develop strategies to exploit them, rather than suppressing them.

One of the first proposals to use Brownian motion of signals in computation originates with Bennett [4]. It takes the form of a mechanical Turing machine, in which signals move around randomly, searching their way through the machine's circuit topology. Later proposals have employed fluctuations with the eye of making a trade-off between energy use and reliability [16, 11], but these approaches tend to require extensive error correction, and may thus fall in the realm of more-or-less traditional methods. Noise and fluctuations have also been used in the simulated annealing process of a Boltzmann machine implemented by Single Electron Tunneling electronics [22]. This method utilizes fluctuations to search in an energy landscape, thus showing some similarities with the Brownian search in Bennett's Turing machine. It is, however, focused on neural networks, rather than on arithmetics-based computation.

This paper presents an asynchronously timed CA in which computation is realized through cell configurations that implement circuits in which signals fluctuate randomly in forward and backward directions, as if they were subject to Brownian motion. The fluctuations are employed in a search process through computational state space that eventually culminates into signals appearing at designated output terminals, which then indicates the termination of the computation. Though the randomness of the fluctuations looks just like that—randomness—it actually forms a powerful resource that can be employed to backtrack circuits out of deadlocks and to equip the circuits with arbitration ability. The fluctuations add to the functionality of the circuits to the extend that only two very simple primitive modules suffice as the basis from which more complicated circuits can be constructed. As a result, the CA requires merely three states and three transition rules, which is far less than achieved thus far in literature for computationally universal asynchronous CA models (see for example the asynchronous CA in [18], which requires six transition rules as well as cells with many states).

Brownian CA have the potential to become the basis for architectures of nanocomputers. Not only are they governed by a very small number of transition rules—thus offering the possibility of simple physical realizations—they also behave somewhat similar to natural phenomena, in which Brownian motion plays a major role. This is particularly clear in biological systems [5, 23]. Molecular motors, for example, use Brownian motion to generate movement that require only tiny amounts of energy—a feat unparalleled by the macro-scale man-made motors based on current technology. If electronic circuits can exploit fluctuations with a similar efficiency, they may be able to operate closer to the thermal limit, which delineates the areas at which signal values can still be discriminated between. Operation closer to the thermal limit may result in the consumption of decreased amounts of power—an important issue in VLSI that will only grow in importance as feature sizes of electronic devices decrease.

This paper is organized as follows. Section 2 introduces the concept of Brownian circuits, because they are the basis from which the Brownian CA is constructed. Two primitive modules are introduced, and it is shown that they form a universal set of primitives from which any arbitrary Brownian circuit can be constructed. The Brownian CA model based on these two primitive modules is presented in Section 3, and some basic configurations are shown. More complicated configurations are shown in Section 4. We finish this paper with Conclusions and a Discussion in Section 5.

2 Brownian circuits: the Hub and the Conservative Join

Brownian circuits are token-based. *Tokens* are discrete indivisible units that are used as signals in circuits and systems. A typical example of token-based systems are *Petri-nets* (e.g. see [7]), which are commonly used for modeling circuits and systems. Token-based circuits are especially promising for physical implementations in which signals have a discrete character. *Single Electron Tunneling (SET)* circuits [15] form an example, as well as *molecule cascades* [10], which consist of individual molecules moving one by one into lower-energy states, and which are triggered by each other, like domino stones.

The token-based circuits considered in this paper are *Delay-Insensitive*, which means that such circuits are robust to any finite delays of signals with respect to the correctness of their output. A delay-insensitive circuit allows arbitrary delays of its signals anywhere in the circuit—be it in wires or in operators—and the delays do not compromise the operation of the circuit. Delay-insensitive circuits are not governed by a clock, so they are part of the class of *asynchronous* circuits [8].

Like synchronously timed circuits, which can be constructed from a limited set of primitives (e.g. NOT-gates and AND-gates), delay-insensitive circuits can also be constructed from a fixed set of primitives, be it a different one. An example of universal primitive elements for delay-insensitive circuits is the set consisting of the so-called *Merge*, *Conservative Tria (CTria)* and *Conservative Sequencer (CSequencer)* (see Fig. 1). These primitive elements have in common that they *conserve* tokens. In other words, the number of input tokens equals the number of output tokens in each primitive. The class of delay-insensitive circuits that conserve tokens is called *Conservative Delay-Insensitive (CDI)* [17].

When the movements of tokens in wires and modules fluctuate between going forward and backward, we say that the tokens undergo Brownian Motion. Circuits that have such fluctuating tokens are called *Brownian Circuits* [20, 13]. This paper assumes that Brownian circuits conserve their tokens. Since we are in the framework of delay-insensitive circuits, any delays to tokens due to Brownian motion will not affect the correctness of a circuit's operation. The frequent undoing of operations may make the overall efficiency of the circuit worse, though.

The added value of allowing Brownian motion of tokens is that, due to the reverse movements in a Brownian circuit, it is possible to undo transitions, including those that lead to deadlocks. Deadlocks may occur in token-based delay-

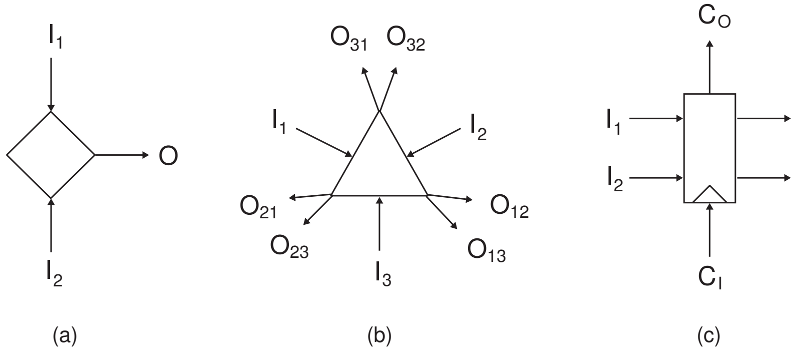


Fig. 1. Primitive elements for delay-insensitive circuits that conserve tokens. (a) Merge merges two streams of inputs I_1 and I_2 into one output stream O. (b) CTria joins two input tokens resulting in two output token as follows. Upon receiving an input signal from each of the two wires I_i ($i \in \{1,2,3\}$) and I_j ($j \in \{1,2,3\} \setminus \{i\}$), it outputs a signal to the wires $O_{6-i-j,i}$ and $O_{6-i-j,j}$. If there is only a signal on one of the input wires, it is kept pending, until a signal on one more input wire appears. (c) CSequencer facilitates arbitration of shared resources between parallel processes. An input signal on wire I_1 (resp. I_2) together with an input signal on wire C_I but without an input signal on wire I_2 (resp. I_1) are assimilated, resulting in an output signal on wire O_1 (resp. O_2) and on wire C_O. If there are input signals on both I_1 and I_2 at the same time as well as an input signal on C_I, then only one of the signals on I_1 and I_2 (possibly chosen arbitrarily) is assimilated together with the signal on C_I, resulting in an output signal on the corresponding O_1 or O_2 wire and on wire C_O. The remaining input signal will be processed at a later time, when a new signal is available on wire C_I.

insensitive circuits, and it usually requires special functionality in the circuits to resolve them, but this causes an increased complexity in primitives and circuits. Random fluctuations of tokens can provide this functionality as part of their nature, thus allowing for simpler primitives and circuits.

How simple can the primitives for Brownian circuits be? The two building blocks used in this paper can be shown to form a computationally universal set from which a wide variety of CDI circuits can be constructed [13].

The first building block is the *Hub*, which contains three wires that are bidirectional (Fig. 2). There will be at most one signal at a time on any of the Hub's wires, and this signal can move to any of the wires due to its fluctuations.

The second building block is the *Conservative Join (CJoin)*, which has two input wires and two output wires, all bi-directional (Fig. 3). The CJoin can be interpreted as a synchronizer of two signals passing through it. Signals may fluctuate on the input wires, and when processed by the CJoin, they will be placed on the output wires where they may also fluctuate. The operation of the

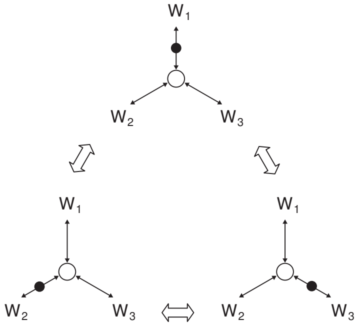


Fig. 2. Hub and its possible transitions. A token is denoted by a black blob. Fluctuations cause a token to move between any of the Hub's three wires W_1, W_2, and W_3 in any order.

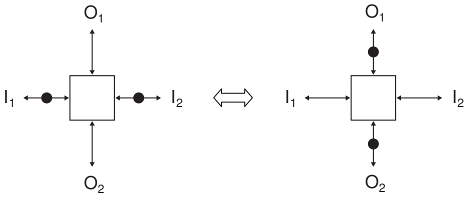


Fig. 3. CJoin and its possible transitions. If there is a token on only one input wire (I_1 or I_2), this token remains pending until a signal arrives on the other wire. These two tokens will then result in one token on each of the two output wires O_1 and O_2.

CJoin may also be reversed, and the forward / backward movement of the two signals through it may be repeated an unlimited number of times. Due to this bidirectionality, there is strictly speaking no distinction between the input and output wires to the CJoin, though we still use the terminology of input and output, since the direction of the process is eventually forward.

It is possible to construct the modules in Fig. 1 from the Hub and the CJoin. The Merge follows directly from the Hub, and is thus trivial so no figure is given for it. The constructions for the CTria and the CSequencer are given in Figs. 4 and 5. Search based on random fluctuations plays an important role in the designs for the CTria and the CSequencer. In the CTria in Fig. 4, for example, a signal input to I_3 is unaware of whether the other required input signal to the CTria will be to I_1 or to I_2, as a result of which both possibilities need to be checked. Since the I_3 input signal will fluctuate between the two corresponding CJoins (at the lower half of the circuit in Fig. 4), it will only be processed by the one CJoin that has a second input signal available. In other words, the two input signals to a CTria search for each other according to a random (Brownian) process, and when they happen to be at the input terminals of one and the same CJoin at a certain time, they will be operated upon by the CJoin. As long as the two input signals fail to find each other at a CJoin, no operations by any CJoin will take place. A similar search process of input signals for a "matching" CJoin takes place in the CSequencer in Fig. 5, where the input signal from C_I fluctuates between the two CJoins. In conclusion, the three modules in Fig. 1 can be constructed from Hub and CJoin modules, and this underlines the universality of the Hub and the CJoin for the class CDI.

There is yet another module that operates on tokens, but that is not strictly necessary to achieve universality. This module is the *Ratchet*, which restricts the

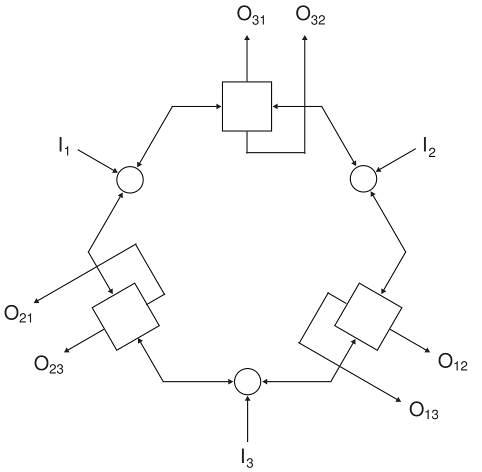


Fig. 4. CTria constructed from Hubs and CJoins

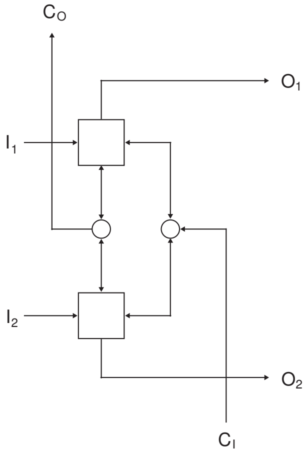


Fig. 5. CSequencer constructed from Hubs and CJoins

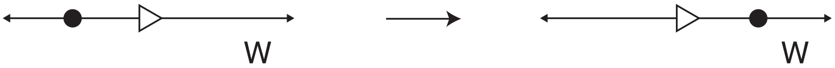


Fig. 6. Ratchet and its possible transition. The token on the wire W may fluctuate before the ratchet as well as after the ratchet, but once it moves over the ratchet it cannot return. The ratchet thus imposes a direction on a (originally) bi-directional wire.

movement of tokens through itself to one direction. It acts as a kind of diode, thus effectively transforming a bidirectional wire into a unidirectional wire (see Fig. 6). The ratchet is used to speed up searching in circuits. Since searching is unable to backtrack over a ratchet, it will consume less time as a result. Use of the ratchet, however, comes at a price: it cannot be placed at positions that interfere with the search process in a circuit, so its use should be carefully considered.

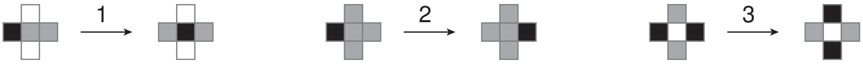


Fig. 7. The three transition rules of the Brownian Cellular Automaton. Rule 1 describes signal propagation on a wire, Rule 2 describes signal propagation over a wire crossing, and Rule 3 describes the processing of two signals by a CJoin.

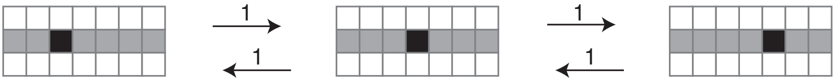


Fig. 8. Wire with a signal on it, and a typical sequence of transitions governing the Brownian motion of the signal. The labels on the arrows indicate which of the three rules is applied.

3 The Brownian cellular automaton model

The Brownian CA model used in this paper is based on a 2-dimensional cell space, in which cells have one of three states, encoded by gray scales. White (state 0) indicates the background used in the cell space, gray (state 1) indicates the wires and operators laid out on the cell space, and black (state 2) indicates signals. The model assumes a von Neumann neighborhood for the cells, i.e. the states of the north, south, west, and east neighbors of a cell are taken into account in the cell's update, as well as the state of the cell itself. The cells are updated according to three transition rules, which are shown in Fig. 7.

The rules are rotation symmetric, meaning that their equivalents rotated by multiples of 90 degrees are also transition rules. Unlike transition rules in traditional CA with von Neumann neighborhood, the rules change the state of not only a cell, but may also do so for its four neighbors.

Updates of the cells take place asynchronously, in the way outlined in for example [2]: at each time step one cell is selected randomly from the cell space as a candidate to undergo a transition, with a probability between 0 and 1. If the state of a selected cell and the states of its neighboring cells match the states in the Left-Hand-Side of a transition rule, the corresponding transition is carried out.

To implement CDI circuits on the cell space, we first describe some basic configurations. We begin with a wire and a signal on it (Fig. 8). A signal has no direction, and due to Rule 1 it can fluctuate forward and backward. This characteristic, which has no counterpart in CA models presented to date, results in an extremely simple representation of a signal, in which no distinction is made between a signal's head and its tail. This is one of the important factors contributing to the small number of states and the small number of transition rules in the Brownian CA model.

Curves are used to change directions of signals (see Fig. 9), and movement forward and backward on them is driven by Rule 2. Both left curves and right curves can be constructed in this way.

The fluctuations of signals place a large time overhead on the computation process, and to deal with that, ratchets are placed at strategic positions in circuits. They guarantee a consistent forward bias of a fluctuating signal toward the output terminals of the circuit, as we have seen in Section 2. The configuration for a ratchet is described in Fig. 10.

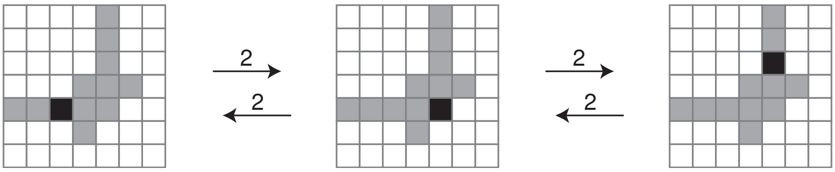


Fig. 9. Curve with a signal on it, and a typical sequence of transitions governing the Brownian motion of the signal.

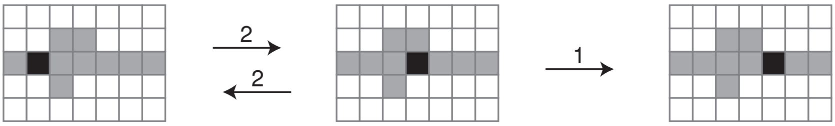


Fig. 10. Wire with a ratchet and a signal on it, and a typical sequence of transitions governing the Brownian motion of the signal. After the signal undergoes the transition by Rule 1, it cannot go back.

Most circuits have some wires that cross each other, and the implementation of this on asynchronous CA models requires some arbitration mechanism to decide which one of the two signals on a crossing may pass the crossing first. On an asynchronous CA, arbitration can be implemented through a specialized circuit, like in [12], or through equipping signals themselves with arbitration functionality through additional transition rules, like in [1]. Brownian CA models, on the other hand, have a much simpler mechanism: the Brownian motion of signals allows them to arbitrate their crossings on their own, without there being special functionality required (Fig. 11). When the transition rule for crossing signals (i.e. Rule 2) does not apply due to both signals being at the crossing, the Brownian motion of the signals will eventually move one of them backwards, away from the crossing, thus opening the way for the second signal to cross, after which the first signal will also be able to cross.

The implementation of the Hub on the cell space is based on Rule 2 (Fig. 12). A signal may fluctuate between all three wires of the Hub, with the central cell being used to temporarily hold the signal in an intermediate location. A 4-wire version of the Hub is also possible, and we leave its design as an exercise to the reader.

Finally, we have the cell configuration for the CJoin in Fig. 13. This configuration relies on Rule 2 to place signals near the center of the CJoin, after which Rule 3 conducts the CJoin operation.

4 Implementations of the CTria and the CSequencer

Having established the cell configurations for the CDI circuit elements, we can now construct the configurations for the CTria and the CSequencer on the cell

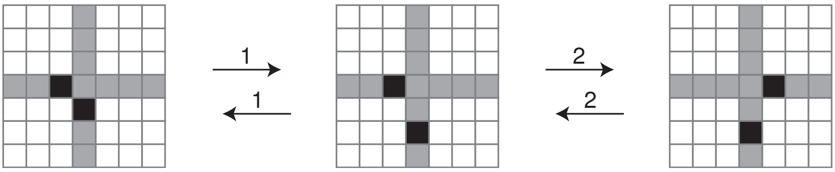


Fig. 11. Crossing wires with two signals on them trying to cross each other, and a typical sequence of transitions governing the Brownian motion of the signals. Initially the two signals cannot cross, since they block each other (left), but the Brownian motion to which they are subject will inevitably move one of them backward (center), freeing the way for the other to cross (right). Eventually, both signals will have moved over the crossing, though this may require many iterations of the signals moving forward and backward.

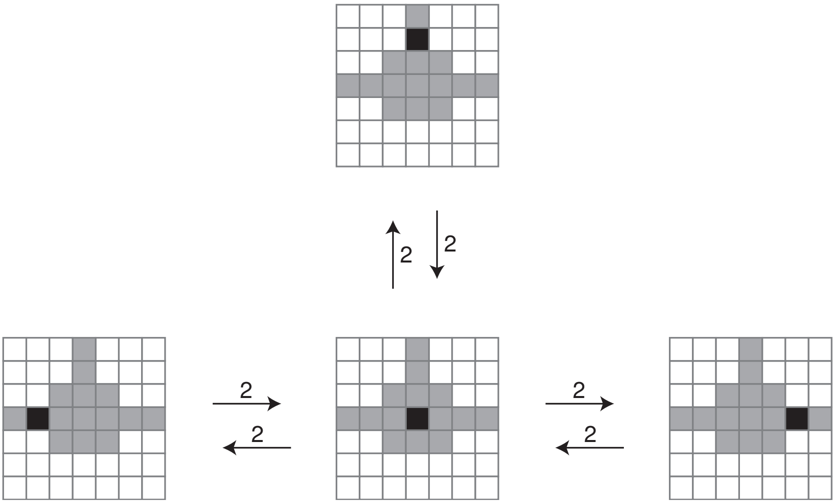


Fig. 12. Hub with a signal on it, and a typical sequence of transitions governing the Brownian motion of the signal. The signal may fluctuate between the three wires of the Hub.

space. The CTria in Fig. 14 is a straightforward design, which closely reflects the circuit in Fig. 4. More compact designs are possible, but they would be less educational, so we leave them out here. The input and output wires of the CTria in Fig. 14 are all equipped with ratchets in order to decrease the search time of the signals inside the circuit. Once in the circuit (i.e. beyond the ratchets at the input wires), input signals search for a matching CJoin in the circuit; they are not allowed to go back through the ratchets, which significantly restricts their search space and, related to it, reduces their search time. Upon being processed and output by a CJoin, the signals pass the ratchets at the output wires, after

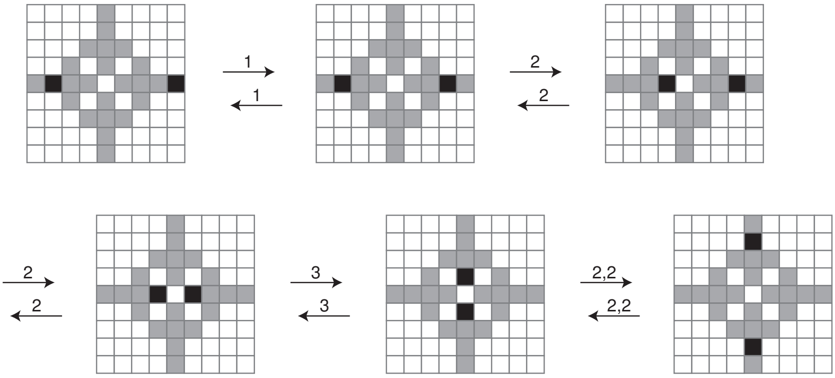


Fig. 13. CJoin with two signals being input to it, and a typical sequence of transitions governing the operation of the primitive. Both signals need to be present for Rule 3 to apply.

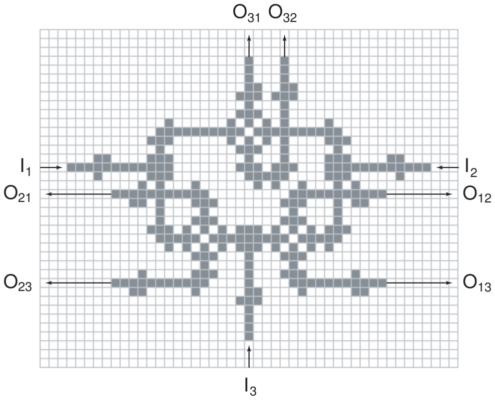


Fig. 14. Construction of the CTria from Hub and CJoin modules on the cell space

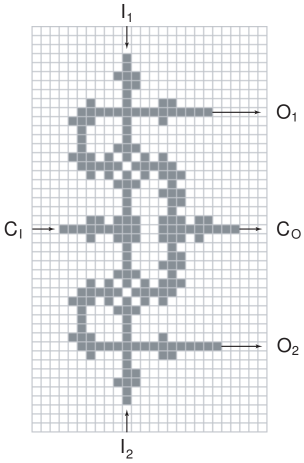


Fig. 15. Construction of the CSequencer from Hub and CJoin modules on the cell space

which they are unable to return to the inside of the circuit. Again, this reduces the search time.

The CSequencer is designed with the same philosophy in mind. Its topology closely reflects the circuit in Fig. 5, but, like with the CTria, educational value is the first priority, not compactness. Ratchets are again placed at the input and output wires to reduce the search times of the signals in the circuit.

5 Conclusions and discussion

Noise and random fluctuations are increasingly considered factors limiting the pace at which integrated electronics develops, prompting some researchers to investigate whether they can be exploited in ways resembling those in biological systems [5]. We have presented a model that uses Brownian motion as a resource to search through computational state space in token-based asynchronous circuits, and have implemented a novel CA based on such circuits. The resulting Brownian CA is remarkably simple, underlining the power of Brownian motion when used as a resource in computation.

Brownian motion-based computing schemes may find applications in future nanocomputing designs in which signals have a discrete and undivided character, such as Single-Electronic Tunneling technology [15]. Preliminary research has already confirmed the feasibility of designs of the Hub and the CJoin based on this technology through computer simulations [21].

The Brownian CA model proposed in this paper uses transition rules in which a cell's state is updated in a transition that simultaneously changes the states of the cell's four neighbors, making the model different than the usual von Neumann-type CA. Though the simultaneous update of a cell and its four neighbors in a single operation may seem to require sophisticated circuitry in the implementation of the CA, there is evidence that such an update rule may actually have its counterparts in molecular systems [3]. Physical couplings between neighboring cells may underlie the mechanisms through which such locally synchronized interactions take place.

The asynchronous nature of the update process in the proposed Brownian CA may be physically more plausible than the globally synchronous updating mechanisms employed in traditional CA models, as it is difficult to keep all cells synchronized in large cell spaces [19]. Some improvements in the proposed update method may still be required, however, since it may be just as difficult to restrict the update of randomly selected cells to a single cell at a time, as is done in the model. Ideally, updates of cells should be completely random, without any restrictions other than that cell states match the states in the Left-Hand-Sides of transition rules—an update method used in [12].

This paper shows that, notwithstanding the randomness associated with Brownian motion, useful computation can still be conducted on the CA model. Computations are subject to search processes, however, and an important question then becomes how much time will be required for searching. It is well-known that the mean displacement of a particle through Brownian motion is proportional to the square root of time [6]. We may thus expect that a token on one end of a wire of length L will require a time of the order $O(L^2)$ before it reaches the other end of the wire, if the token is subject to Brownian motion. Though this is a significant overhead, it can be limited by keeping the distances short over which Brownian motion acts on tokens. In other words, the placement of ratchets on wires at preferably constant-order distances from each other will significantly speed up computations, since the time required for a token to propagate over a wire then reduces to a linear order. Most circuits will probably allow such an arrangement; for example, the configurations for the CTria and the CSequencer modules in Fig. 14 and 15, respectively, have ratchets at all the input wires as well as at all the output wires, and the distances between the input and output ratchets is relatively short. Consequently, though search takes place in both these modules, the processes are not very time-consuming. Simulations conducted on a computer have confirmed this.

References

[1] S. Adachi, J. Lee, and F. Peper. On signals in asynchronous cellular spaces. *IEICE Trans. inf. & syst.*, E87-D(3):657–668, 2004.

[2] S. Adachi, F. Peper, and J. Lee. Computation by asynchronously updating cellular automata. *Journal of Statistical Physics*, 114(1/2):261–289, 2004.

[3] A. Bandyopadhyay and S. Acharya. A 16-bit parallel processing in a molecular assembly. *Proceedings of the National Academy of Sciences*, 105(10):3668–3672, March 2008.

[4] C.H. Bennett. The thermodynamics of computation—a review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.

[5] S. Dasmahapatra, J. Werner, and K.-P. Zauner. Noise as a computational resource. *Int. J. of Unconventional Computing*, 2(4):305–319, 2006.

[6] A. Einstein. *Investigations on the Theory of the Brownian Movement*, translation of "Annalen der Physik, 17, pp. 549–560, 1905", pages 1–18. Dover Publications, New York, 1956.

[7] C. Girault and R. Valk. *Petri Nets for Systems Engineering.* Springer, 2003.

[8] S. Hauck. Asynchronous design methodologies: an overview. *Proc. IEEE*, 83(1):69–93, 1995.

[9] Simon Haykin. *Adaptive filter theory (3rd ed.).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[10] A.J. Heinrich, C.P. Lutz, J.A. Gupta, and D.M. Eigler. Molecule cascades. *Science*, 298:1381–1387, 2002.

[11] L.B. Kish. Thermal noise driven computing. *Applied Physics Letters*, 89(14):144104–1–3, 2006.

[12] J. Lee, S. Adachi, F. Peper, and K. Morita. Embedding universal delay-insensitive circuits in asynchronous cellular spaces. *Fundamenta Informaticae*, 58(3/4):295–320, 2003.

[13] J. Lee, F. Peper, et. al. Brownian circuits — Part II: Efficient designs and brownian cellular automata. *In preparation*, 2008.

[14] K. Morita, Y. Tojima, K. Imai, and T. Ogiro. *Universal computing in reversible and number-conserving two-dimensional cellular spaces*, chapter "Collision-based computing", pages 161–199. Springer-Verlag, London, UK, 2002.

[15] Y. Ono, A. Fujiwara, K. Nishiguchi, H. Inokawa, and Y. Takahashi. Manipulation and detection of single electrons for future information processing. *Journal of Applied Physics*, 97:031101–1–031101–19, 2005.

[16] K.V. Palem. Energy aware computing through probabilistic switching: a study of limits. *IEEE Trans. Computers*, 54(9):1123–1137, 2005.

[17] P. Patra and D.S. Fussell. Conservative delay-insensitive circuits. In *Workshop on Physics and Computation*, pages 248–259, 1996.

[18] F. Peper, J. Lee, F. Abo, T. Isokawa, S. Adachi, N. Matsui, and S. Mashiko. Fault-tolerance in nanocomputers: a cellular array approach. *IEEE Transaction on Nanotechnology*, 3(1):187–201, 2004.

[19] F. Peper, J. Lee, S. Adachi, and S. Mashiko. Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers? *Nanotechnology*, 14(4):469–485, April 2003.

[20] F. Peper, J. Lee, et. al. Brownian circuits — Part I: Concept and basic designs. *In preparation*, 2008.

[21] S. Safiruddin, S.D. Cotofana, F. Peper, and J. Lee. Building blocks for fluctuation based calculation in single electron tunneling technology. In *Proceedings of the 8th International Conference on Nanotechnology (IEEE Nano)*, 2008.

[22] T. Yamada, M. Akazawa, T. Asai, and Y. Amemiya. Boltzmann machine neural network devices using single-electron tunnelling. *Nanotechnology*, 12(1):60–67, 2001.

[23] T. Yanagida, M. Ueda, T. Murata, S. Esaki, and Y. Ishii. Brownian motion, fluctuation and life. *Biosystems*, 88(3):228–242, 2007.

On coupling random Boolean networks

Larry Bull[1] and Ramón Alonso-Sanz[1,2]

[1] Department of Computer Science, University of the West of England, Bristol
Frenchay Campus. Bristol BS16 1QY, UK
[2] ETSI Agrónomos (Estadística), C.Universitaria. 28040, Madrid, Spain

{Larry.Bull, Ramon.Alonso-Sanz}@uwe.ac.uk

Abstract. Traditional Random Boolean networks (RBN) consist of nodes within a single network, each updating synchronously, although asynchronous versions have also been presented. In this paper the dynamics of multiple, mutually coupled traditional RBN are investigated. In particular, the effects of varying the degree of inter-network connectivity and differing intra-network connectivity are explored. The effects from different inter-network evolution rates are then considered, i.e., asynchronousity at the network level is examined. Finally, state memory is included within the nodes of coupled RBN and shown to alter the dynamics of the networks under certain circumstances.

1 Introduction

Random Boolean networks (RBN) [16] were originally introduced to explore aspects of biological genetic regulatory networks and can be viewed as a generalization of binary cellular automata (CA) [23]. Since then they have been used as a tool in a wide range of areas such as self-organisation (e.g., [17]), computation (e.g., [10]), robotics (e.g., [21]) and artificial creativity (e.g. [9]). Traditional RBN consist of N nodes, each connected to K other randomly chosen network members, with each performing a randomly assigned Boolean update function based on the current state of those K members in discrete time steps. Such updates are typically executed in synchrony across the network but asynchronous versions have also been presented (after Harvey and Bossomaier [13]), leading to a classification of the space of possible forms of RBN [11]. All of this work has assumed a single network. However, many systems of interest may be viewed as consisting of multiple networks, each with their own internal structure and "coupling structure" to their external world partners. Examples include economic markets, social networks, ecologies, and within organisms such as neural-immune networks. Following Morelli and Zanette [19], in this paper the dynamics of multiple, coupled RBN are explored with a view to presenting an abstract tool with which to explore the dynamics of such systems. The initial results presented here indicate that the relative rate of network evolution is the most significant factor in determining network behaviour. Similarly, the inclusion of a memory mechanism such that updates are not made based purely upon the current state of network nodes but rather upon an historical view considering their previous states can also significantly alter network behaviour.

2 Random Boolean networks

As noted above, within the traditional form of RBN presented by Kauffman, a network of N nodes, each with K connections to other nodes in the network, all update synchronously based upon the current state of those K nodes. Since they have a finite number of possible states and they are deterministic, such networks eventually fall into a basin of attraction. It is well-established that the value of K affects the emergent behaviour of RBN wherein attractors typically contain an increasing number of states with increasing K. Three phases of behaviour are suggested: ordered when $K{=}1$, with attractors consisting of one or a few states; chaotic when $K > 3$, with a very large numbers of states per attractor; and, a critical regime around $K{=}2$, where similar states lie on trajectories that tend to neither diverge nor converge and 5-15 % of nodes change state per attractor cycle (see [17] for discussions of this critical regime, e.g., with respect to perturbations). Figure 1 shows examples of ten randomly created $N{=}1000$ networks, each started from 10 random initial configurations, for varying K. The fraction of nodes which change state per update cycle is recorded and shown in the figures. Thus it can be seen that this number is typically very low for low K but increases rapidly with $K > 2$, as expected.

3 Coupled RBN

Kauffman also recast RBN as a model of fitness adaptation for evolving species — the *NK* model — whereby each node represents a gene and its fitness contribution is based upon K other genes, each possible configuration being assigned a random value. It is shown how K effects fitness landscape topology. Later, Kauffman made the point that species do not evolve independently of their ecological partners and presented a coevolutionary model of fitness adaptation [18]. Here each node/gene is coupled to K others locally and to C within each of the S other species with which it interacts — the *NKCS* model. Each species within such models experiences constant changes in the shape of its fitness landscape due to the evolutionary advances of its coupled neighbours until mutual equilibria are reached. The *NKCS* model has been used to explore a number of aspects of natural coevolution, such as symbiosis [e.g., [7]] and collective behaviour [e.g., [6]], along with coevolutionary adaptation in general [e.g., [4]] and coevolutionary computation [e.g., [5]]. A similar extension can be made to the RBN framework such that $S{+}1$ networks exist, wherein each node is connected both to K randomly chosen nodes within its own network and C randomly chosen nodes in each of the S other networks. However, unlike the radical change from the *NK* model to the *NKCS* model, the effects of adding C connections appears roughly equivalent to increasing K in a single network. That is, given

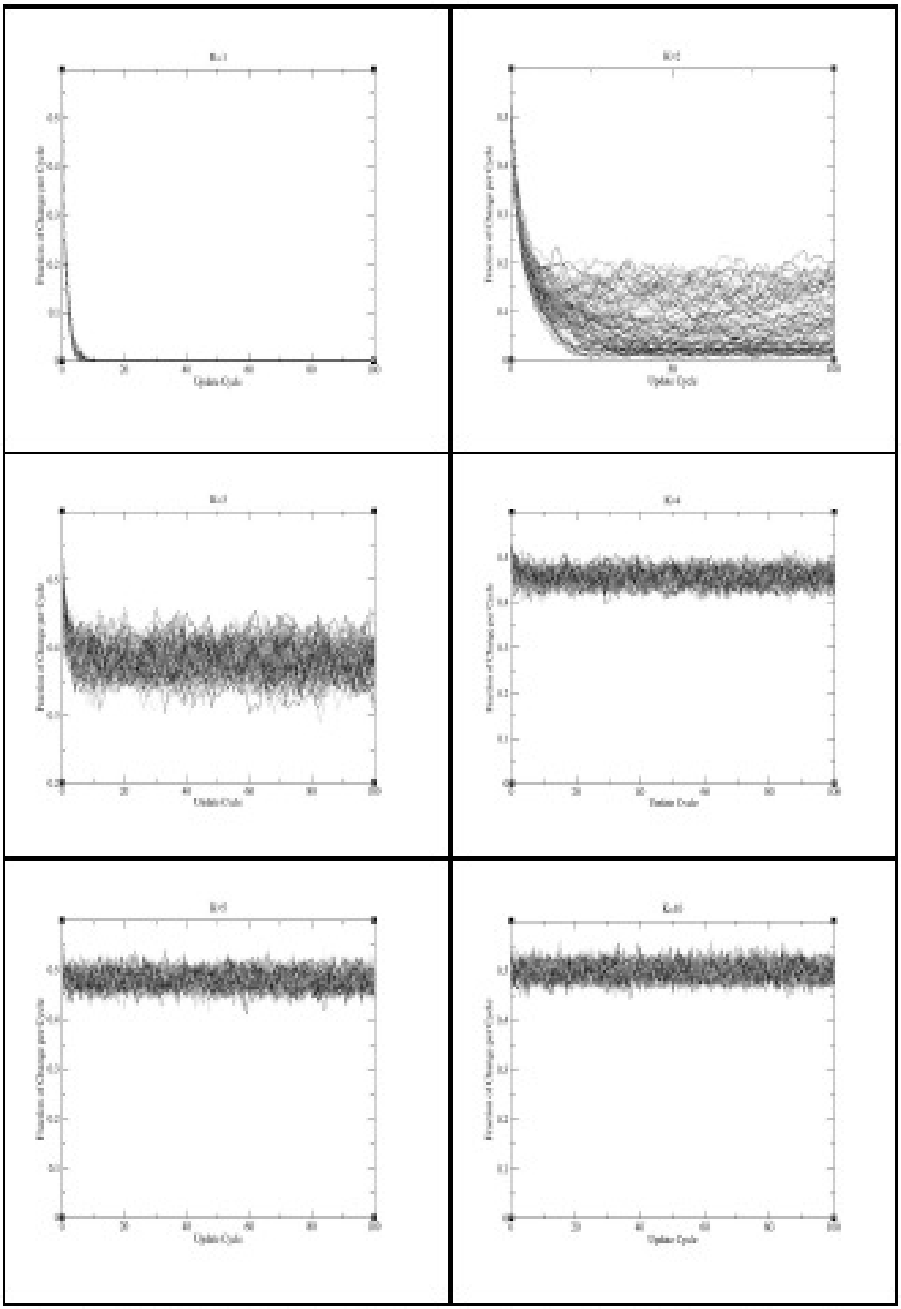


Fig. 1. N=1000 RBN and various K, showing fraction of nodes which change state per step.

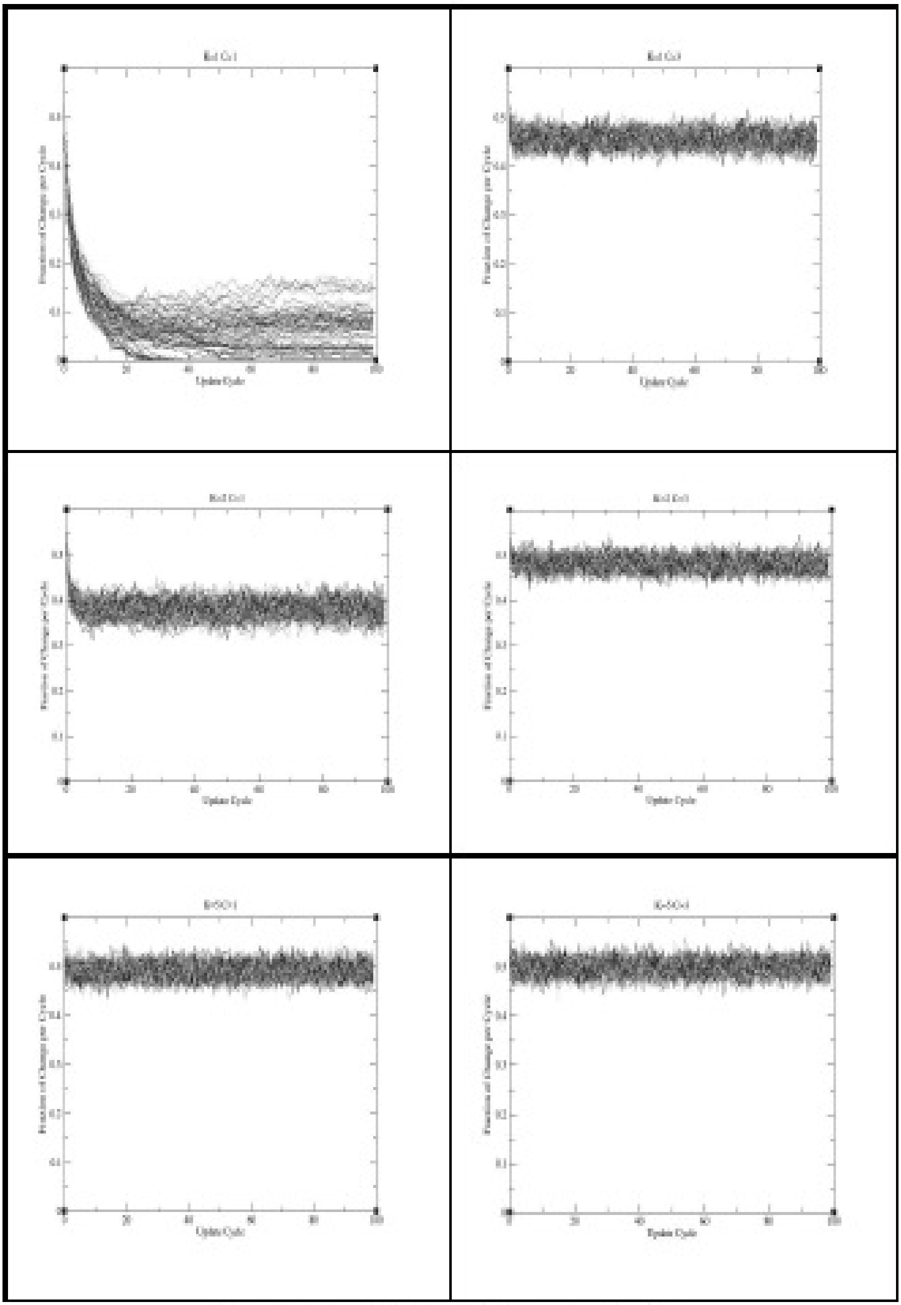


Fig. 2. Two N=1000 RBN, with various K and C.

that all connections are randomly assigned and all updates are synchronous, any of the $S+1$ networks simply behave as if they have $K + (S \times C)$ connections within a standard RBN, as shown in Fig. 2 for two networks of the type used above. It should be noted that the global network is now of size $(S+1)N$. Morelli and Zanette [19] used probabilistic connections between two RBN, wherein they explored synchronisation between the networks with K=3 (see Villani et al. [22] for a related study).

4 Non-Symmetrical K

After Morelli and Zanette [19], Hung et al. [14] examined two coupled disordered cellular automata with probabilistic connectivity of varying K using elementary rule 22. They report a change in the synchronisation dynamic when the internal coupling K is different in each network. Indeed, in the broader view, there is no reason to assume that the degree of connectivity within a given network, i.e., its basic internal structure, will be the same as that of its coupled partner network(s). Kauffman and Johnsen [18] also showed changes in behaviour due to heterogeneous values of K for the coevolving populations of the *NKCS* model. In particular, they describe how low K fitness increases for high C when partnered with a high K species before equilibrium but that only low K fitness increases under low C.

Figure 3 shows examples of how under low inter-network coupling, i.e., $C=1$, the differing intra-network coupling has an effect on dynamics. Here the lower K partner (for $K < 4$) experiences an increase in the fraction of nodes which change state per update cycle compared to the symmetrical case and the higher K partner (for $K > 1$) experiences a slight decrease despite the overall increase in connectivity. The effects are lost on the higher K partner when C is increased.

5 Coupled RBN with different evolution rates

In all of the aforementioned examples of multiple-network scenarios it is likely that in many cases some networks will be changing at different rates relative to each other: technological networks may change faster than ecological ones, for example. Therefore a new parameter R can be introduced which specifies the number of update cycles a given network undertakes before another performs one update cycle. Thus the updates of individual networks become deterministically asynchronous, somewhat akin to deterministic asynchronous single network RBN [11]. In these RBN, nodes update if a defined number of cycles have passed since their last update. It is known that within single network deterministic asynchronous RBN the amount of nodes updated per cycle and the frequency of such updates can significantly change the overall behaviour despite the connectivity and update rules remaining constant [12]. Figure 4 shows how this can also be true within multiple coupled RBN, in this case for two networks of the type used above. Indeed, for low K, as R increases, it can be seen that network behaviour exhibits a phase transition-like phenomenon such that the effects of

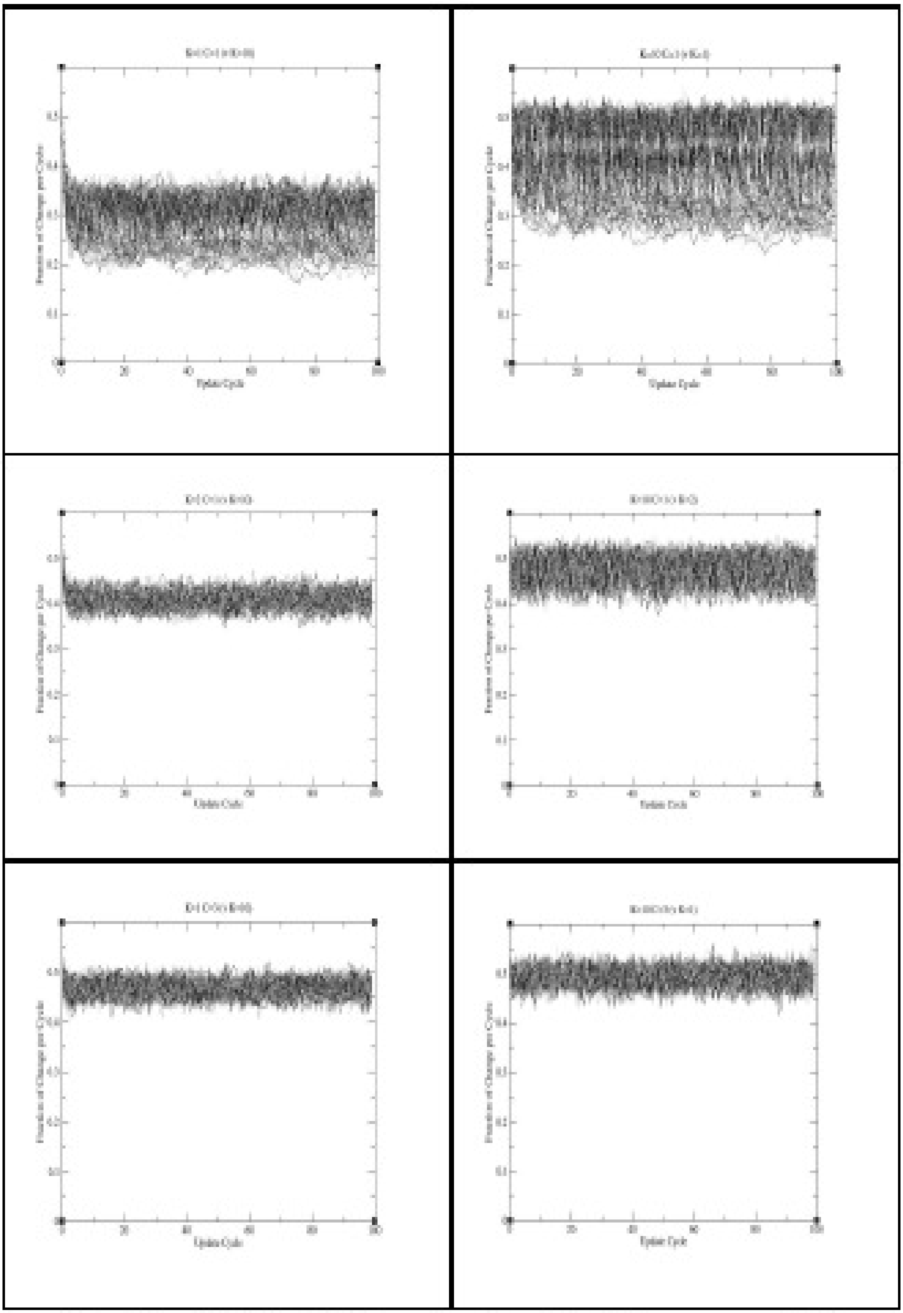


Fig. 3. Two N=1000 RBN, with example pairings of K and C for non-symmetrical K.

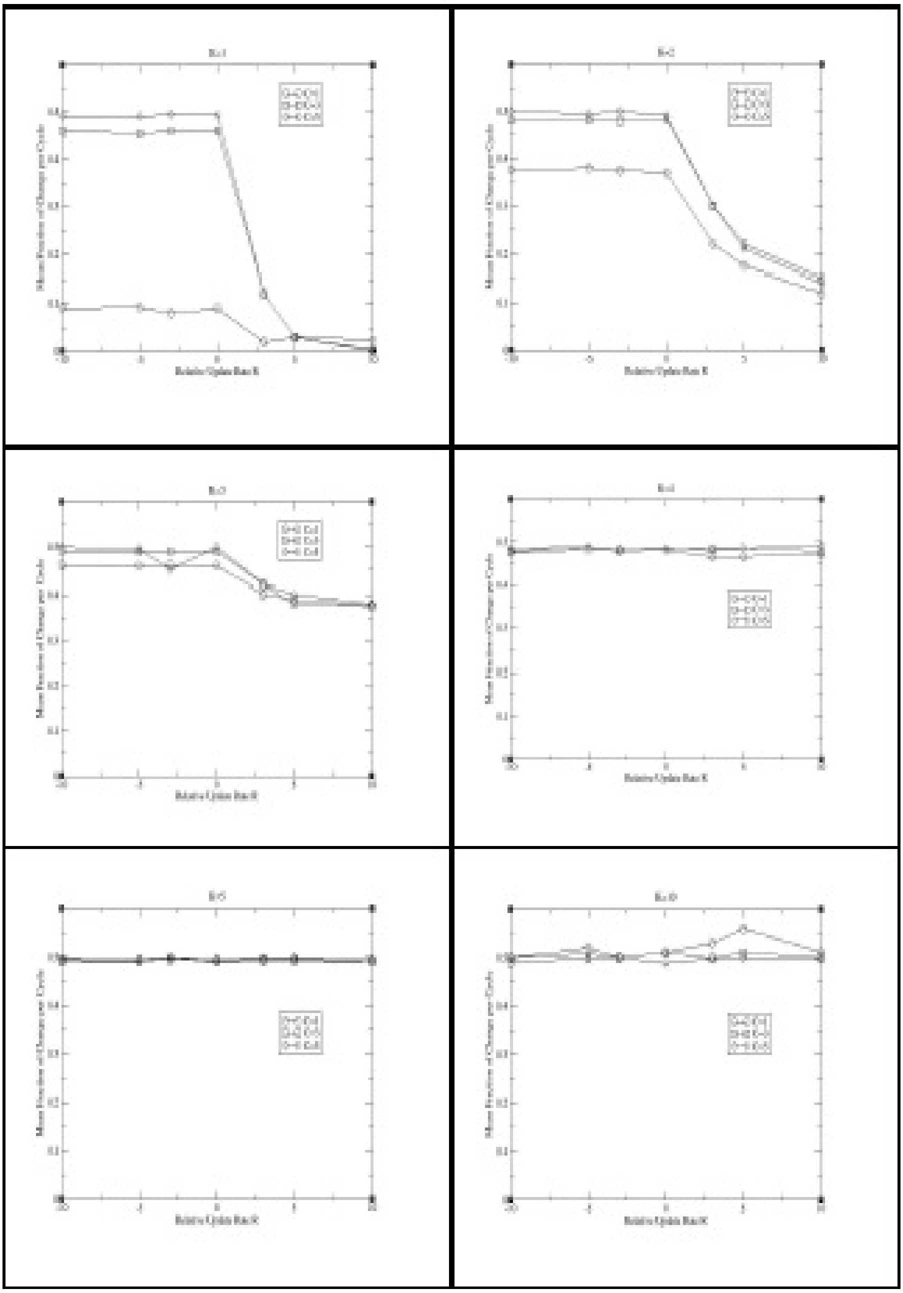


Fig. 4. Two N=1000 RBN, with various K, C and R. Mean fraction of change after 100 cycles per configuration. Note a negative number implies a slower relative rate R, e.g., -10 is when a network updates 10 times slower than the other network.

the inter-network coupling C are increasingly lost. That is, for a given intra-network connectivity K, as R increases, the fraction of nodes which change state per update cycle reduces to those seen in the traditional uncoupled RBN case (compare to Fig. 1). The relative rate required to reach the equivalent single network behaviour increases with K. However, for networks with predominantly chaotic behaviour, i.e., $K > 3$, no change in behaviour is seen from varying R for any C, as might be predicted.

6 Memory

In standard RBN the update occurs based on the current state of the K nodes: no previous node state is explicitly considered. Alonso-Sanz and Cárdenas [2] have recently implemented a simple form of state memory within $K{=}4$ RBN such that nodes use a weighted mean value of previous state to apply the update rules to. They describe how such memory has a dampening effect wherein the fraction of nodes changing state per update cycle is reduced. A similar mechanism has been explored within the multiple network RBN. Here a memory mechanism is implemented such that the total historic average state of a node is used for updates, with this memory parameter m maintained using the well-known Widrow-Hoff Delta rule with learning rate β: $m^{t+1} = m^t + \beta(s^t - m^t\ )$. Where s^t is the current state of the node, $\beta{=}0.1$, m^0 is 0.5 and thereafter the input to the Boolean functions considering the node is said to be logical '1' when $m^t > 0.5$ and logical '0' otherwise. Figure 6 shows how, using symmetrical K networks as in Fig. 2 and 4 , a similar general inertia effect as reported by Alonso-Sanz and Cárdenas [2] can be seen. As in Fig. 4 , increasing R decreases the fraction of nodes which update per cycle down to the same fraction obtained without inter-network coupling C (and with memory). These rates of change are always lower than for the equivalent non-memory networks. Note also the effect shows for all K tried, unlike in Fig. 4. Further, the dynamics of networks which are slower than their partner(s) are significantly altered by the inertia effect for lower K. Here, as R decreases, the fraction of nodes changing state per update increases towards the level of networks with high overall coupling, i.e., $K + (S \times C)$, and no memory, as shown in Fig. 4 . That is, the sharp phase transition-like effect seen without memory is dampened by the memory mechanism.

This dampening effect is somewhat unexpected since the inclusion of memory can be seen as increasing the degree of coupling within RBN — coupling is in both time and space. A minimal memory mechanism has also been explored within RBN [3] here such that updates are based on the current and previous state of nodes only, i.e., an explicit memory of one time-step. Thus, in the single network case, for a given K there exists 2^{2K} possible states per node. Figure 5 shows the effects of this simple memory on a single network and how, for $K > 1$, the behaviour for a given K is indeed like that found for the equivalent $2K$ network (compare with Fig. 1). In the $K{=}1$ case, the time taken to reach an attractor is increased but the fraction of nodes which update per cycle does not increase. It therefore appears that within the ordered regime the degree of

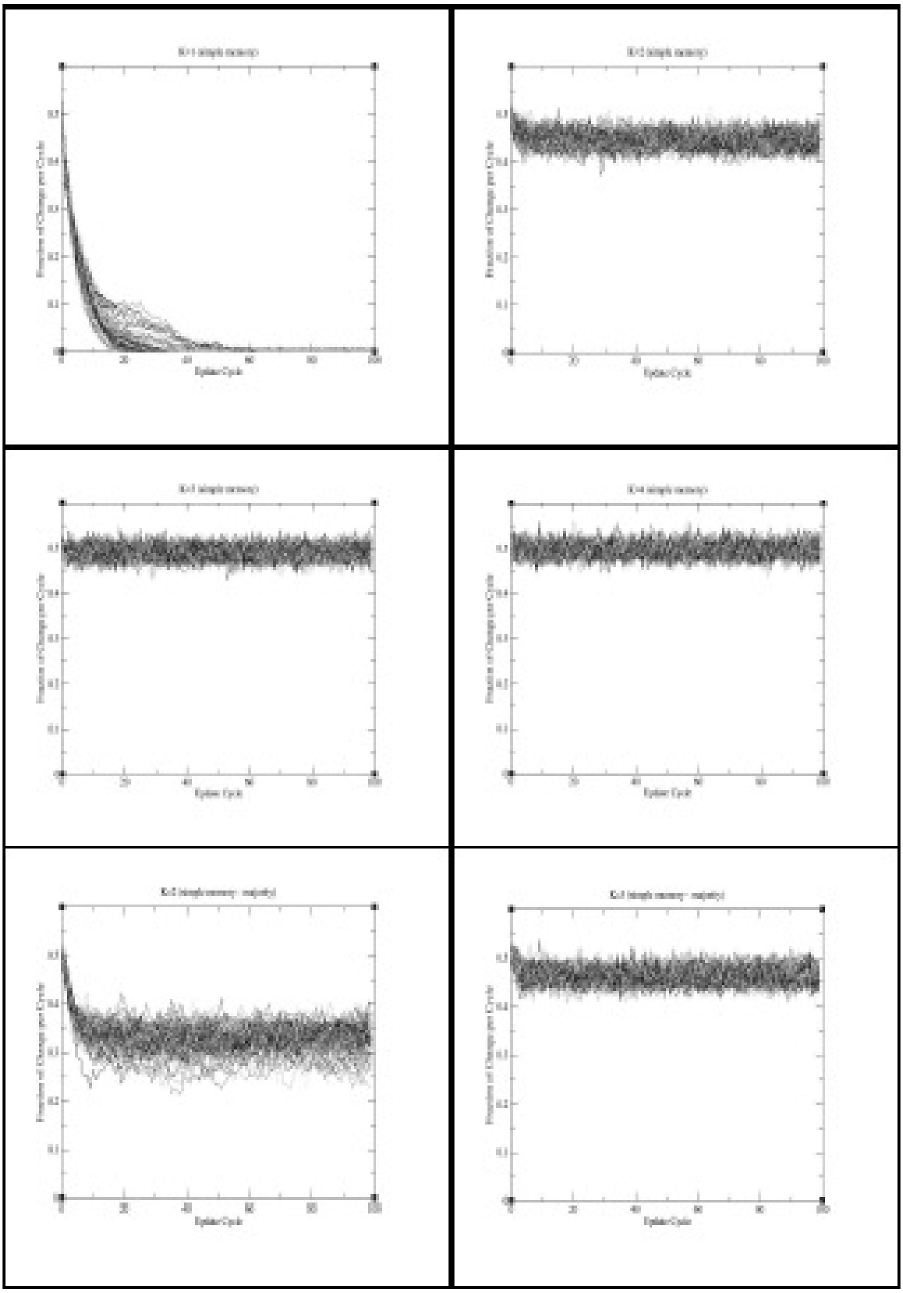


Fig. 5. The effects of a minimal memory mechanism of one time-step for various K in a single RBN. Also showing the effects from altering how the previous step is considered.

typical isolation of the nodes is such that the increase in temporal connectivity has no significant effect on attractor length. The last two results in Fig. 5 show how altering the simple mechanism such that the previous node state is only considered if both the current and previous states are logical 1 (i.e., mode) reduces the effects of the temporal coupling. The memory mechanism of Alonso-Sanz and Cardenas [2], like that used in Fig. 6, is a majority vote over the total previous states. It therefore seems that the longer the period over which this sampling of logical 1 activity is taken, the greater the reduction of the effects of the coupling in time and, perhaps more surprisingly, in space also.

7 Conclusions

This paper has explored a recently introduced extension of the RBN framework: multiple, coupled networks. Such multi-network systems represent a way in which to model a variety of systems of interest, both artificial and natural. It has been confirmed that synchronously updated networks behave in roughly the same way as single networks when the total connectivity in the former is equal to that in the latter, i.e., when $K + (S \times C) = K^{single}$. An effect of coupling is to increase the overall N of the global network which may be significant since the typical time taken to reach an attractor and the number of states per attractor increases with N [17]. However, previous work exploring the effects of enforcing a topological structure within traditional RBN with scale free sub-networks (e.g., [20],[1],[24]) has suggested that the number of attractors and their length can be greatly decreased in comparison to the uniform case. The same appears to be true here since, for example, there is no significant increase in the time to reach attractors between the single RBN in Fig. 1 and their equivalent in terms of connectivity in Fig. 2. It has been shown that when the internal connectivity of the coupled networks is different the dynamics are changed such that, for example, high K networks coupled to low K networks experience a drop in the typical amount of state change per cycle in comparison to their uncoupled level, i.e., despite an increase in overall coupling. When the updating is asynchronous, a phase transition-like phenomenon is seen for low intra-network connectivity wherein the effects of inter-network coupling are essentially removed for sufficiently different rates of updating. Finally, the effects of state memory have been explored. Results confirm previous reports of a dampening effect, which is shown to alter the aforementioned phase transition-like phenomenon such that it exists for networks with higher intra-network connectivity since the inertia keeps them out of the chaotic phase of RBN.

A number of analytical results with respect to the number of states within an attractor have been presented for traditional RBN: Median number $= 0.5 \times 2N/2$ for $K{=}N$ (e.g. [17]); Median number $= \sqrt{N}$ for $K{=}2$ [8]; and, Median number $= \sqrt{\pi}/2\sqrt{N}$ for $K{=}1$ [15]. The equivalent $K{=}N$ case is when $K + (S \times C) = (S+1)N$. Given that full connectivity removes any topology effects, the median number will be $0.5 \times 2^{((S+1)N)/2}$. With non-symmetrical K values, the relative difference in the K values of the partners must be considered. In the

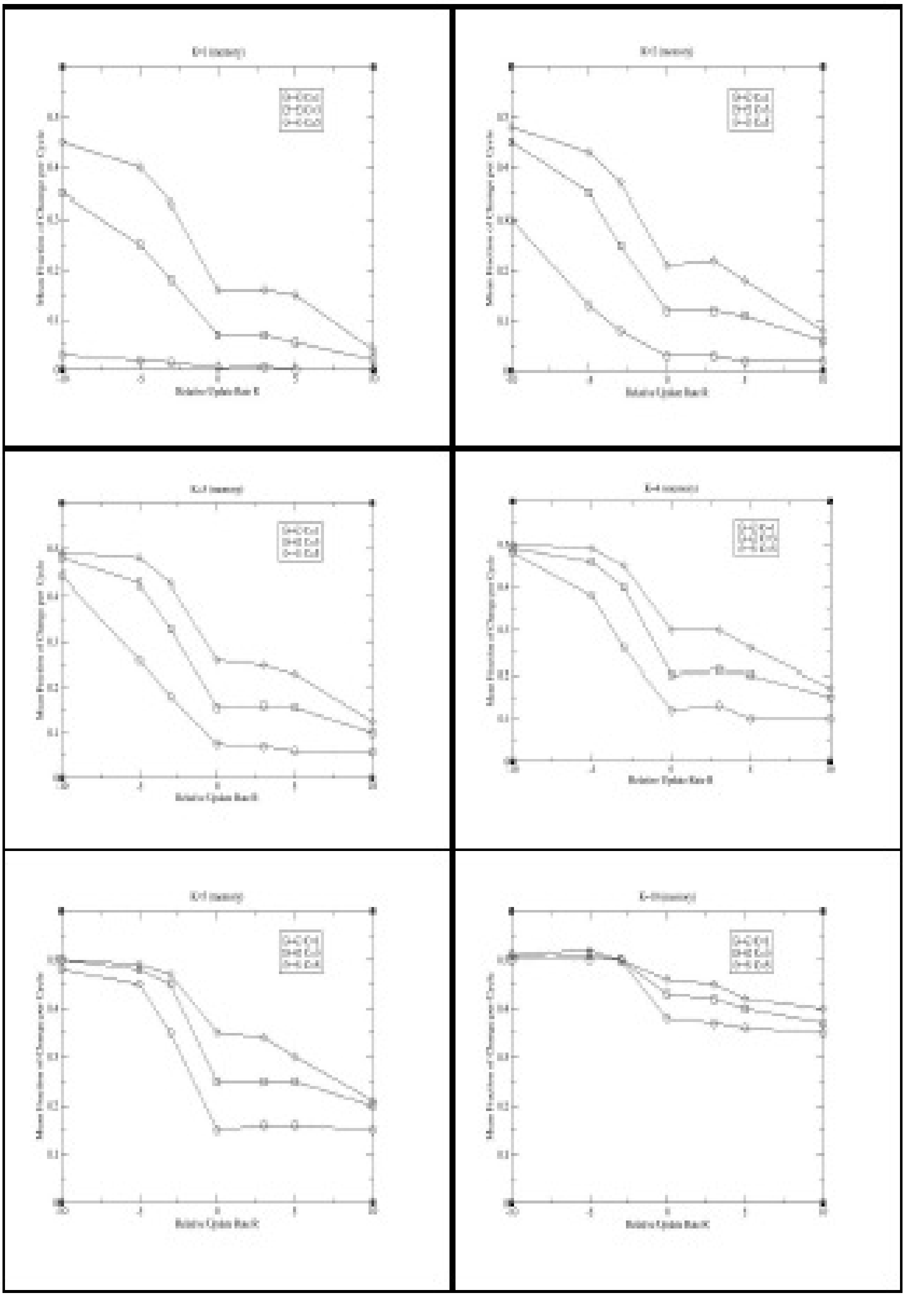


Fig. 6. Two N=1000 RBN, with various K, C and R with state memory. Mean fraction of change after 100 cycles shown per configuration for each network.

asynchronous case, it seems the relative update rate R proportionally reduces the effects of the $S \times C$ inter-network connections. Future work will seek to explore both aspects formally. Other areas of immediate interest include varying network topologies, using asynchronous RBN of various types, and exploring other forms of memory mechanism and producing analytical results from their inclusion.

Other forms of coupled discrete dynamical system are currently being explored.

Acknowledgements

This work was supported under EPSRC grant no. EP/E049281/1.

References

[1] Aldana, M.(2003) Boolean dynamics of networks with scale-free topology. Physica D 185(1): 46-66.

[2] Alonso-Sanz, R. and Cardenas, J.P. (2007) Effect of memory in Boolean networks with disordered dynamics: the K=4 case . Int. J. Modern Physics C 18(8): 1313-1327.

[3] Alonso-Sanz, R and Bull, L. (2008) Boolean networks with memory. Int. J. Bifurcation and Chaos (in press).

[4] Bak, P., Flyvbjerg, H. and Lathrup, B.(1994) Evolution and coevolution in rugged fitness landscapes. In C. Langton (Ed.) Artificial Life III. Addison-Wesley, 11-42.

[5] Bull, L.(1997) Evolutionary computing in multi-agent environments: partners. In Proceedings of the Seventh International Conference on Genetic Algorithms. Morgan Kaufmann, 370-377.

[6] Bull, L.(1999) On the evolution of multicellularity and eusociality. Artificial Life 5(1):1-15.

[7] Bull, L. and Fogarty, T.C.(1996) Artificial symbiogenesis. Artificial Life 2(3):269-292.

[8] Derrida, B. and Pomeau, Y.(1986) Random networks of automata: a simple annealed approximation. Europhys. Lett. 1(2): 45-49.

[9] Dorin, A.(2000) Boolean networks for the generation of rhythmic structure. In Proceedings of the Australian Computer Music Conference, 38-45.

[10] Fernández, P. and Solé, R.(2004) The role of computation in complex regulatory networks. In E. Koonin, Y. Wolf & G. Karev (Eds.) Power Laws, Scale-free Networks and Genome Biology. Landes.

[11] Gershenson, C.(2002) Classification of random Boolean networks. In R.K. Standish, M. Bedau & H. Abbass (Eds.) Artificial Life VIII. MIT Press, 1-8.

[12] Gershenson, C, Broekaert, J. and Aerts, D.(2003) Contextual random Boolean networks. In Proceedings of the Seventh European Artificial Life Conference. Springer, 615-624.

[13] Harvey, I. and Bossomaier, T.(1997) Time out of joint: attractors in asynchronous random Boolean networks. In Proceedings of the Fourth European Artificial Life Conference. MIT Press, 67-75.

[14] Hung, Y-C., Ho, M-C., Lih, J-S. and Jiang, I-M.(2006) Chaos synchronisation in two stochastically coupled random Boolean networks. Physics Letters A 356: 35-43.

[15] Jaffee, S.(1988) Kauffman ntworks: cycle structure of random clocked Boolean networks. Ph. D. Thesis. New York University.

[16] Kauffman, S.A.(1969) Metabolic stability and epigenesis in randomly constructed genetic nets. Journal of Theoretical Biology 22:437-467.

[17] Kauffman, S.A.(1993) The Origins of Order: Self-Organization and Selection in Evolution. Oxford.

[18] Kauffman S.A. and Johnsen, S.(1991) Coevolution to the edge of chaos: coupled fitness landscapes, poised states, and coevolutionary avalanches. In C. Langton (Ed.) Artificial Life III. Addison-Wesley, 325-370.

[19] Morelli, L.G. and Zanette, D.H.(2001) Synchronisation of Kauffman networks. Physical Review E 63.

[20] Oosawa, C. and Savageau, M.(2002) Effects of alternative connectivity on behaviour of randomly constructed Boolean networks. Physica D 170: 143-161.

[21] Quick, T., Nehaniv, C., Dautenhahn, K. and Roberts, G.(2003) Evolving embedded genetic regulatory Network-driven control systems. In Proceedings of the Seventh European Artificial Life Conference. Springer, 266-277.

[22] Villani, M., Serra, R., Ingrami, P. and Kauffman, S.A.(2006) Coupled random Boolean networks forming an artificial tissue. In Proceedings of the Seventh International Conference on Cellular Automata for Research and Industry. Springer, 548-556.

[23] Von Neumann, J.(1966) The Theory of Self-Reproducing Automata. University of Illinois.

[24] Wuensche, A.(2004) Basins of attraction in network dynamics: a conceptual framework for biomolecular networks. In G. Schlosser and G.P. Wagner. (Eds) Modularity in Development and Evolution. Chicago University Press, 288-311.

Cellular automata with dynamically reconfigurable buses

Thomas Worsch

Universität Karlsruhe, Fakultät für Informatik
worsch@ira.uka.de

Abstract. We consider one-dimensional cellular automata which are extended by dynamically reconfigurable buses (RCA). It is shown that up to a constant factor it does not matter whether bus segments are directed or not. For both variants of the model their time complexity can be characterized in terms of the reversal complexity of one-tape TM. The comparison of RCA with tree CA shows that the former are in the second machine class and that they can be transformed in some normal form with an only polynomial overhead of time.

1 Introduction

Dynamically reconfigurable architectures have received growing attention during the last years. A lot of different models have been investigated. They all have in common that some kind of processors are exchanging information on global buses the structure of which can be modified in each step by each processor by segmenting and/or fusing bus segments locally. It is the characteristic of a bus that information travels over arbitrary distances in constant time.

Bus automata were one of the first models introduced by Rothstein [9]. While it used finite automata, most of the models considered later [1, 2, 5, 6, 10, 11] differ in two respects: processors are RAM with a memory of non-constant size and the models are two- or even higher-dimensional.

In this paper we are interested in the case of one-dimensional systems of finite automata with reconfigurable buses which we call *reconfigurable cellular automata* for short. Even though such systems do not have a large bisection width, they are in the second machine class.

This paper is an abridged version of [13]. It is organized as follows. In Sect. 2 reconfigurable cellular automata are introduced. In Sect. 3 polynomial time simulations between RCA and tree CA (TCA) are established and their consequences are discussed. In Sect. 4 the relation of (both variants of) RCA to sequential TM is investigated.

2 Basic notions

A *reconfigurable cellular automaton* with k bus segments ($k-$RCA) consists of a one-dimensional array of finite automata (cells) working synchronously according to the same local rule. Q denotes the set of states of one cell and B the bus alphabet. Cells are connected by bus segments. In the case of bidirectional bus segments there are k such segments between each two immediately neighboring cells and we write $\mathrm{R_2CA}$. In the case of unidirectional bus segments there are k bus segments for information transmission from cell i to $i+1$ and k segments in the opposite direction. The model is denoted as $\mathrm{R_1CA}$.

One step of each cell consists of four phases:

1. **Configure:** Depending on the local state, arbitrary subsets of (locally available) bus segments are connected.
2. **Send:** Depending on the local state, on each such subset the cell may send a symbol $b \in B$ or send nothing.
3. **Receive:** On each bus segments the cell may observe a symbol sent by a cell participating in the bus.
4. **New state:** Depending on the old state and the symbols received (if any) the cell enters a new state.

In the case of $\mathrm{R_2CA}$ a symbol sent spreads along all bus segments belonging to the bus on which it was sent. In an $\mathrm{R_1CA}$ a symbol only spreads along the directions of the segments. One can distinguish different conflict resolution strategies for write conflicts on a bus. In all constructions sketched below, one can always avoid such conflicts algorithmically.

3 RCA are in the second machine class

In this section $\mathrm{R_2CA}$ will be shown to belong to the second machine class, i.e. the problems solvable by them in polynomial time are exactly those in PSPACE. This will be done by describing polynomial time simulations of and by tree-shaped CA (TCA). That trees as an underlying topology of one kind or the other are a powerful tool has been shown by several authors. Wiedermann [12] was probably the first; see also [4, 8].

Theorem 3.1. *Let t be a function satisfying the requirements of Lemmata 3.1 and 3.2. Then:*

$$\mathrm{R_2CA{-}TIME(Pol}(t)) = \mathrm{TCA{-}TIME(Pol}(t)) \ .$$

Remark 3.1. It immediately follows that $\mathrm{RCA{-}TIME(Pol}(n))$ =PSPACE, i.e. $\mathrm{R_2CA}$ are in the second machine class.

For the proof of the theorem one can use the following two lemmata.

Lemma 3.1. *Let t be a function such that a* TCA *can mark the cells in a subtree of height* $\mathrm{O}(t)$*. Then:*

$$\mathrm{R_2CA{-}TIME}(t) \subseteq \mathrm{TCA{-}TIME(O}(t^3)) \ .$$

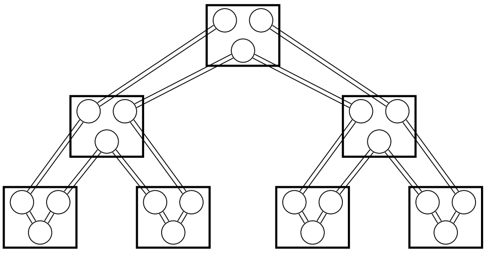


Fig. 1. Mapping cells of an R_2CA (circles) to cells of a TCA (rectangles).

Sketch of proof. The cells of an R_2CA R are mapped to cells of a TCA T as indicated in Fig. 1. The simulation of one step of R is done in a number of steps proportional to the square of the height of the tree. Since that is the logarithm of the maximum number of cells used by R, it needs time $O(t^2)$.

(Technical details will be provided in a full version of the paper.) □

Lemma 3.2. *Let* t *be a function such that an* R_2CA *can mark* 2^t *cells in time* $O(t^2)$. *Then:*

$$\text{TCA}-\text{TIME}(t) \subseteq R_2\text{CA}-\text{TIME}(O(t^2)) \ .$$

Sketch of proof. The cells of a TCA T are mapped to cells of an R_2CA R as indicated in Fig. 2. The simulation of one step of T is done in a number of phases. Each phase takes a constant number of steps and is used to simulate the exchange of data between two successive layers of cells of T. Thus the overall time for the simulation of one step of T is proportional to the height of T. Since the height of the used part of the tree can be bounded by the number of steps of T, the total running time of R is at most $O(t^2)$.

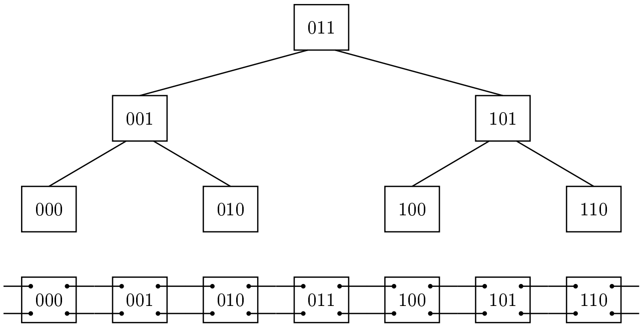


Fig. 2. Mapping TCA cells to R_2CA cells for a tree of height 2.

Details of the construction which are concerned with technicalities as for example segmenting the buses at the right places, handling of inputs, and marking of certain segments of cells of R are omitted. □

Corollary 3.1. *Given an arbitrary* $\mathrm{R_2CA}$ R *one can use Lemmata 3.1 and 3.2 to construct a* $\mathrm{R_2CA}$ R' *(via a* TCA*) which simulates* R *in polynomial time and has certain "nice" properties:*

1. *In* R' *there are only* 2 *bus segments between neighbors (instead of* k*).*
2. *All buses of* R' *are linear (while buses of* R *may fork or form cycles).*
3. *In* R' *only exclusive sends happen (while in* R *common sends or even conflicts may occur).*

4 A characterization of RCA time by TM reversal

Below we write $\mathrm{T_1TM}$ to denote TM with one work tape, one read-write head on it, and without a separate input tape.

Theorem 4.1. *If* f *satisfies the requirements of Lemma 4.1, then:*

$$\mathrm{R_2CA{-}TIME}(\Theta(f)) = \mathrm{R_1CA{-}TIME}(\Theta(f)) = \mathrm{T_1TM{-}REV}(\Theta(f)) \ .$$

One should observe that (somewhat surprisingly) the characterization is in terms of reversal complexity of TM with only one work tape and without an input tape. Usually this model is considered to be too weak and multi-tape machines are used in connection with reversal complexity [3].

A proof of Theorem 4.1 can be given by establishing 3 set inclusion in a circular way "from left to right". One is trivial. The two other cases are treated in Lemmata 4.1 and 4.2 below.

Lemma 4.1. *Let* f *be a function such that a* $\mathrm{T_1TM}$ *can mark* 2^t *tape squares using* $\mathrm{O}(t)$ *reversals. Then an* f *time-bounded* $\mathrm{R_1CA}$ *can be simulated by an* $\mathrm{O}(f)$ *reversal-bounded* $\mathrm{T_1TM}$*.*

Sketch of proof. One has to show how to simulate one step of an $\mathrm{R_1CA}$ R with a constant number of reversals of a TM T. This requires a trick similar to one which is useful in the (omitted part of) the proof of Lemma 3.1.

□

Lemma 4.2. *An* f *reversal-bounded* $\mathrm{T_1TM}$ *can be simulated by an* $\mathrm{O}(f)$ *time-bounded* $\mathrm{R_2CA}$*.*

Sketch of proof. Let T be a $\mathrm{T_1TM}$. Without loss of generality one may assume that T moves its head in every step. We show how a whole sweep of T between two head reversals can be simulated in constant time by a $\mathrm{R_2CA}$ R.

If only an $\mathrm{R_1CA}$ were required one could use Rothstein's idea [9]: Consider e.g. a sweep from left to right. The symbols of successive tape squares are stored in successive cells of R. For each state q of T there is a bus segment l_q entering from the left neighboring cell and a bus segment r_q leaving to the right. For all q and b, if T upon entering a square with symbol b in state q from the left will leave it to the right in state q' after having changed the symbol to b', then a cell storing b connects l_q with $r_{q'}$.

Then the cell responsible for the tape square on which the last reversal happened sends a signal on the segment r_q corresponding to the state on which T leaves the square. Since the bus segments are unidirectional the signal spreads along exactly one path from left to right, "activating" in each cell exactly that l-segment belonging to the state T is in when entering the corresponding tape square. Depending on the state and its stored symbol each cell can enter a new state representing the new tape symbol.

The direct transfer of this idea to $\mathrm{R_2CA}$ results in a failure. Since bus segments are not directed any longer, in each cell which connects at least two l_{q_1} and l_{q_2} to the same $r_{q'}$ the signal arriving on one of the l-segments will spread "backwards" along the others. As a result it may happen that some cells observe the signal on every bus segments.

But it is possible to find the "wrongly" active bus segments in a constant number of phases, each of which only needs a constant number of steps.

□

5 Summary

It has been shown that one-dimensional RCA are in the second machine class. Their time complexity can be characterized in terms of reversal complexity of one-tape Turing machines. Via simulations by and of TCA one can obtain a normal form of RCA with $k = 2$, only linear buses and exclusive sending on the buses working with a polynomial overhead of time.

One notes that the proof techniques cannot be carried over to the two-dimensional case. This case deserves further investigation, as do restrictions on the maximum allowed bus length or the assumption of a non-constant lower bound for the transmission of information on long buses (in order to make the model more realistic).

References

[1] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster. The power of reconfiguration. *Journal of Parallel and Distributed Computing*, 13(2):139–153, 1991.

[2] Y. Ben-Asher, K.-J. Lange, D. Peleg, and A. Schuster. The complexity of reconfiguring network models. *Information and Computation*, 121:41–58, 1995.

[3] J. Chen. The difference between one tape and two tapes: with respect to reversal complexity. *Theoretical Computer Science*, 73:265–278, 1990.

[4] A. Fellah and S. Yu. Iterative tree automata, alternating Turing machines, and uniform boolean circuits: Relationships and characterizations. In H. Berghel et al., eds., *Proc. Symposium on Applied Computing. Volume 2*, p. 1159–1166, 1992.

[5] M. Maresca. Polymorphic Processor Arrays. *IEEE Transactions on Parallel and Distributed Systems*, 4:490–506, 1993.

[6] R. Miller, V. K. Prasanna-Kumar, D. Reisis, Q. F. Stout. Meshes with reconfigurable buses. In *Proc. Conf. on Advanced Research in VLSI*, p. 163–178, 1988.

[7] J. M. Moshell and J. Rothstein. Bus automata and immediate languages. *Information and Control*, 40:88–121, 1979.

[8] J. Mycielski and D. Niwiński. Cellular automata on trees, a model for parallel computation. *Fundamenta Informaticae*, XV:139–144, 1991.

[9] J. Rothstein. On the ultimate limitations of parallel processing. In P. H. Enslow Jr., ed., *Proc. Int. Conf. on Parallel Processing*, p. 206–212, 1976.

[10] S. Sahni. The DMBC: Architecture and fundamental operations. In *Proc. Int. Conf. on Supercomputing*, p. 60–66, ACM Press, 1995.

[11] J. L. Trahan, R. Vaidyanathan, R. K. Thiruchelvan. On the power of segmenting and fusing buses. *Journal of Parallel and Distributed Computing*, 34:82–94, 1996.

[12] J. Wiedermann. Paralelný Turingov stroj — Model distribuovaného počítača. In J. Gruska, ed., *Distribouvané a parallelné systémy*, p. 205–214. Bratislava, 1983.

[13] Th. Worsch. Cellular Automata with Dynamically Reconfigurable Buses In J. Pavelka et al., eds., *Proc. SOFSEM 1999*, LNCS 1725, pp. 488–496, 1999.

.

Asynchronous logic automata

David A. Dalrymple, Neil A. Gershenfeld, and Kailiang Chen

Massachusetts Institute of Technology

Abstract. We present a class of device, termed Asynchronous Logic Automata (ALA), for practical reconfigurable computing that may be categorized as an asynchronous lattice gas automaton, as a Petri net, as a field-programmable gate array, and as charge-domain logic. ALA combine the best features of each: locality, consistent asynchronous behavior, easy reconfiguration, and charge-carrier conservation. ALA are thus "closer to physics" (at least, the physics used in classical computation) than classical computers or gate arrays with global interconnect and clocking, and may therefore be physically implemented with more desirable properties such as speed, power, and heat dissipation.

1 Introduction

Physics, above the atomic level, is inherently local, and computation, like every other process, relies on physics. Thus, programming models which assume non-local processes, such as data buses, random access memory, and global clocking, must be implemented at a slow enough speed to allow local interactions to simulate the non-local effects which are assumed. Since such models do not take physical locality into account, even local effects are limited to the speed of the false non-local effects, by a global clock which regulates all operations.

In computing today, many observers agree that there is a practical physical speed limit for the venerable von Neumann model (see for instance [1]), and that the bulk of future speed increases will derive from parallelism in some form. Chipmakers are currently working to pack as many processors as they can into one box to achieve this parallelism, but in doing so, they are moving even further from the locality that is necessary for a direct implementation as physics. At the other end of the abstraction spectrum, while sequential programming models can be generalized to use multiple parallel threads, such models are often clumsy and do not reflect the physical location of the threads relative to each other or memory.

In addition, research has long suggested that asynchronous (or "self-timed") devices consume less power and dissipate less heat than typical clocked devices [2]. However, traditional microarchitectures require significant book-keeping overhead to synchronize various functional blocks, due to the nature of their instructions, which must be executed in sequence. Most asynchronous designs to present have derived their performance benefits from clever pipelining and power distribution rather than true asynchrony – known as “globally asynchronous, locally synchronous” design – and often this is not enough to offset the overhead [3].

These shortcomings are accepted because of the tremendous body of existing code written in sequential fashion, which is expected to run on the latest hardware. However, by removing the assumption of backwards compatibility, there is an opportunity to create a new, disruptive programming model which is more efficient to physically implement. In particular, such a model could scale favorably to an arbitrary number of parallel elements, to larger problem sizes, and to faster, smaller process technologies. Potentially, this may have eventual impact across the computing industry, particularly in high-performance computing. In addition, it could conceivably be an enabling technology for the Singularity (see [4]).

In Sect. 2, we describe the Logic CA, a synchronous cellular automaton which is the basis of the ALA. In Sect. 3, we introduce the ALA as a modification of the Logic CA. In Sect. 4, we discuss the relationship to past work, and in Sect. 5 we identify future work.

2 Logic CA

Asynchronous Logic Automata (ALA) are based on an earlier model, a cellular automaton (CA), known as the Logic CA. It may be convenient to understand first the Logic CA, which has closer ties to previous work (e.g. [5]), particularly if the reader is familiar with these types of constructions.

The Logic CA consists of cells with 8 neighbors and 9 bits of state. The state bits are divided into 8 configuration bits and 1 dynamic state bit. The configuration bits are further divided into 2 gate bits which choose among the four allowed Boolean functions ($\{AND, OR, XOR, NAND\}$) and 6 input bits which choose among the 36 possible pairs of (potentially identical) inputs chosen from the 8 neighbors ($\frac{1}{2}\cdot 8\cdot(8-1)+8$). At each time step, a cell examines the dynamic state bit of its selected inputs, performs the selected Boolean operation on these inputs, and sets its own dynamic state to the result.

Mathematically, an instance of the Logic CA can be described as a series of global states S_t ($t\in\mathbb{N}_0$) each composed of local states $s_{t,(i,j)}\in\{0,1\}$ ($i,j\in\mathbb{Z}$) and a set of constant configuration elements

$$\begin{aligned}
c_{(i,j)}\in\mathcal{C} &= \big(\{AND, OR, XOR, NAND\}\times(\{-1,0,1\}^2-\{(0,0)\})^2\big)\\
&= \{AND, OR, XOR, NAND\}\\
&\quad\times\{(1,0),(1,1),(0,1),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1)\}\\
&\quad\times\{(1,0),(1,1),(0,1),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1)\}
\end{aligned}$$

(note that there is a bijection between $\mathcal{C}$ and $\{0,1\}^8$, 8 bits) such that

$$s_{t+1,i,j} = \begin{cases} \text{if } \ (c_{(i,j)})_1 = AND & s_{t,(i,j)+(c_{(i,j)})_2} \wedge s_{t,(i,j)+(c_{(i,j)})_3} \\ \text{if } \ (c_{(i,j)})_1 = OR & s_{t,(i,j)+(c_{(i,j)})_2} \vee s_{t,(i,j)+(c_{(i,j)})_3} \\ \text{if } \ (c_{(i,j)})_1 = XOR & s_{t,(i,j)+(c_{(i,j)})_2} \oplus s_{t,(i,j)+(c_{(i,j)})_3} \\ \text{if } \ (c_{(i,j)})_1 = NAND & \neg(s_{t,(i,j)+(c_{(i,j)})_2} \wedge s_{t,(i,j)+(c_{(i,j)})_3}) \end{cases}$$

Although the Logic CA is useful for many applications, we identified two major problems with it, leading to the development of ALA:

1. **Lack of Reversible/Adiabatic Logic.** The system does not employ conservative logic [6] or adiabatic computing [7], which is necessary to truly represent physical resources.
2. **Global Clock.** The clock is global – clearly a non-local effect. Cellular automata are not fundamentally required to have a global clock to perform universal computation [8, 9].

3 Asynchronous logic automata

We have discovered a new approach, inspired by both lattice-gas theory [10] and Petri net theory [11], that resolves the above problems.

By “lattice gas” we mean a model similar to cellular automata in which the cells communicate by means of particles with velocity as opposed to broadcasted states. Practically, this means that the information transmitted by a cell to each of its neighbors is independent in a lattice gas, where in a cellular automaton these transmissions are identical. By convention, a lattice gas also has certain symmetries and conservation properties that intuitively approximate an ideal gas [12], and in some cases, numerically approximate an ideal gas [13].

Meanwhile, Petri nets are a broad and complex theory; we are primarily concerned with the subclass known as “marked graphs” (a detailed explanation can be found in [14]). In short, a marked graph is a graph whose edges can be occupied at any given time by zero or more tokens. According to certain conditions on the tokens in edges neighboring a node of the graph, the node may be allowed to “fire” (at any time as long as the conditions are met), by performing some operations on the tokens (such as moving a token from one of its edges to another or simply consuming a token from an edge).

Our new approach merges these with the existing Logic CA as follows. We remove the global clock and the bit of dynamic state in each cell, and replace the neighborhood broadcasts with a set of four edges between neighboring cells, each containing zero or one tokens, thus comprising a bit of state (see Fig. 1). Between each pair of cells, in each direction, we have a pair of edges, one to represent a “0” signal, and the other a “1” signal. Note that each pair of edges could be considered one edge which can carry a “0” token or a “1” token. Instead of each cell being configured to read the appropriate inputs, this data is now represented by an “active” bit in each edge. Then, each cell becomes a stateless node (except the gate type) in this graph, which can fire on the conditions that all its active

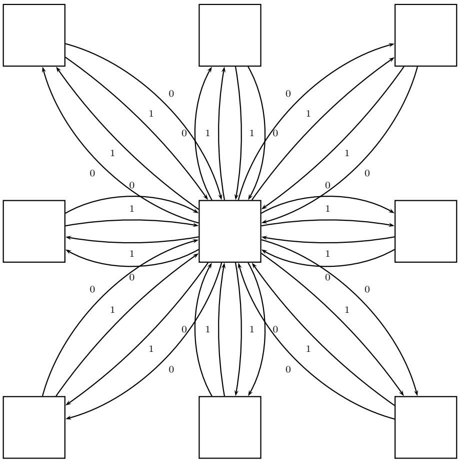


Fig. 1. Edges of one cell in the new approach

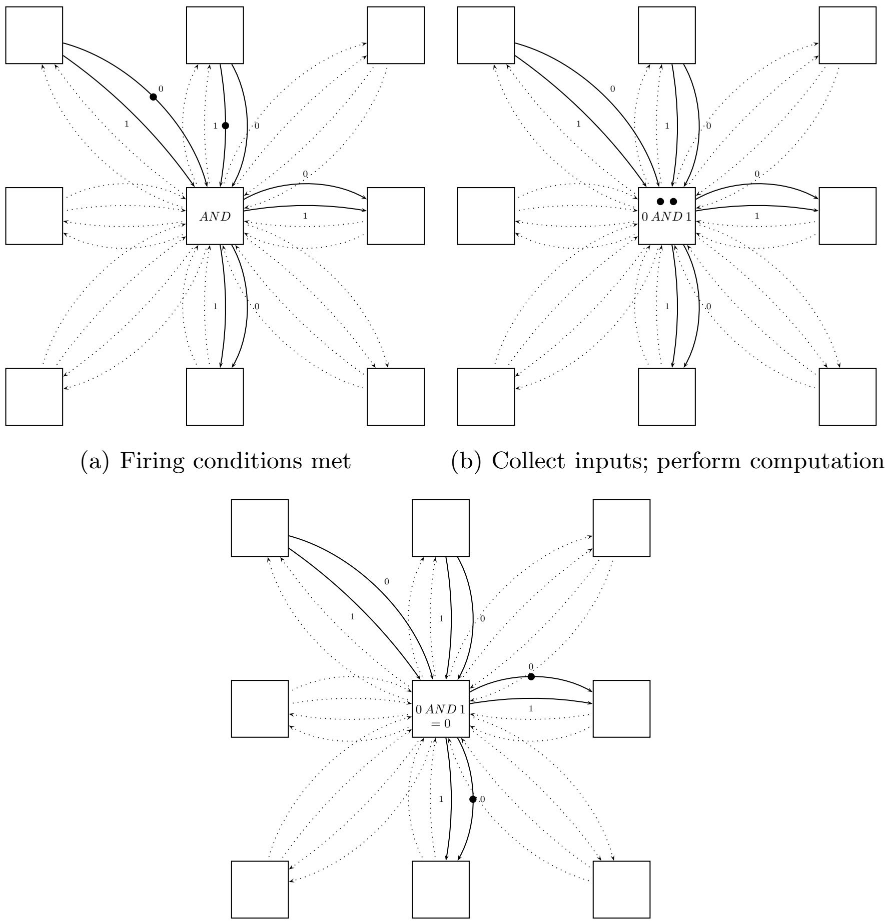
inputs are providing either a “0” token or a “1” token and that none of its active output edges is currently occupied by a token of either type. When firing, it consumes the input tokens, removing them from the input edges, performs its configured function, and deposits the result to the appropriate output edges (see Fig. 2 for an example of a 2-input, 2-output AND gate firing). As it is a marked graph, the behavior of this model is well-defined even without any assumptions regarding the timing of the computations, except that each computation will fire in some finite length of time after the preconditions are met. The model now operates asynchronously, and removes the need not only for a global clock, but any clock at all.

We have also introduced explicit accounting for the creation and destruction of tokens instead of implicitly doing both in every operation, as with traditional CMOS logic. For instance, in Fig. 2, since there are equally many inputs and outputs, no tokens must be created or destroyed. While the model still uses the same irreversible Boolean functions, these functions can be thought of as being simulated by conservative logic which is taking in constants and dispersing garbage [6], enabling an easy pricing of the cost of non-conservatism in any given configuration.

In addition, this model adapts much more easily to take advantage of adiabatic logic design; for instance, when a cell is being used only to ferry tokens from one place to another (e.g. an inverter, shown in Fig. 3), it can do so physically, instead of using a traditional, charge-dumping CMOS stage.

Figures 4 and 5 show the general concept of how such cells could be implemented using so-called “bucket brigade”, or charge-domain logic [15].

Note the following possible ALA variations:



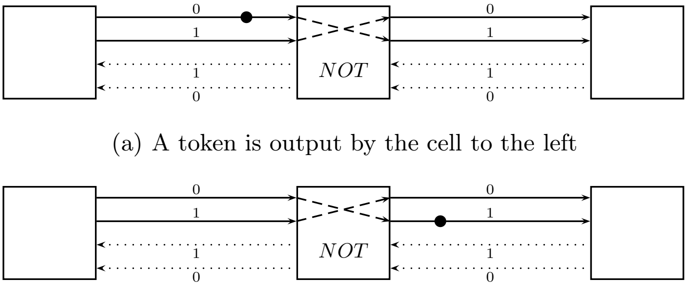
(a) Firing conditions met

(b) Collect inputs; perform computation

(c) Distribute result

Fig. 2. A cell firing; note that despite the loss of information, tokens are conserved in this example

1. **No Diagonals.** Connections may only be present between vertically or horizontally adjacent cells, to simplify circuit layout.
2. **Multiple Signals.** More than four token-storing edges may connect neighboring cells, allowing the conveyance of more parallel information in the same period of time.
3. **More Functions.** The class of possible functions executed by each cell need not be limited to {AND, OR, XOR, NAND} but may include any function $f : \{0, 1, \emptyset\}^n \rightarrow \{0, 1, \emptyset\}^n$ where n is the number of neighbors of each cell (for $n = 8$ there are 43046721 possible functions). A cell executing function f may fire if f's present output is not $\emptyset^n$ and every non-empty element of the



(a) A token is output by the cell to the left

(b) The token now passes to the right cell

Fig. 3. A bit travels left to right through an inverting cell

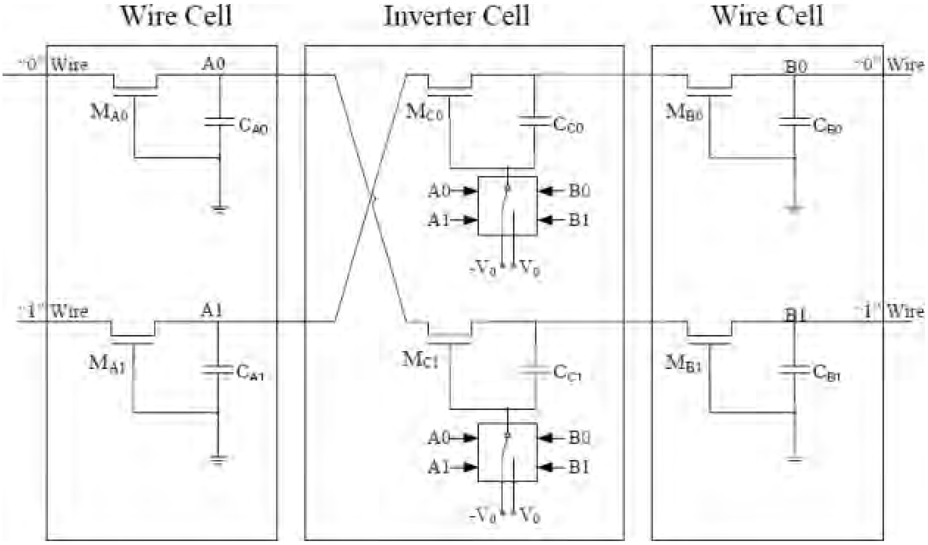


Fig. 4. Transistor-level effects of configured wire and inverter cells

output points to either an inactive or empty set of output edges. Then each of those output edges would become populated with the value specified by f's output. There is a tradeoff between the number of functions allowed and the number of configuration bits in each cell needed to specify the function.

4 History

The history begins with the cellular automata (CAs) of von Neumann [5], designed to explore the theory of self-replicating machines in a mathematical way (though never finished). Note that this was some time after he completed the architecture for the EDVAC project [16], which has come to be known as "the

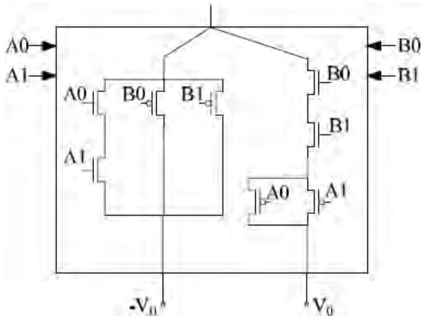


Fig. 5. Expansion of "switch" block from Fig. 4

von Neumann architecture." Many papers since then can be found examining (mostly 2-state) CAs, and there are a few directions to prove simple CA universality – Alvy Ray Smith's [17], E. Roger Banks' [18], and Matthew Cook's more recent Rule 110 construction [19]. However, while interesting from the point of view of computability theory, classical CAs clearly over-constrain algorithms to beyond the point of practicality, except in a small class of problems related to physical simulation (for instance, see [13]).

Another related sub-field is that of field-programmable gate arrays (FPGAs). Gate arrays have evolved over time from sum-product networks such as Shoup's [20] and other acyclic, memory-less structures such as Minnick's [21] to the complex, non-local constructions of today's commercial offerings, yet skipping over synchronous and sequential, but simplified local-effect cells.

The tradition of parallel programming languages, from Occam [22] to Erlang [23] to Fortress [24] is also of interest. Although they are designed for clusters of standard machines (possibly with multiple processors sharing access to a single, separate memory), they introduce work distribution techniques and programming language ideas that are likely to prove useful in the practical application of our work.

Finally, the Connection Machine [25] was designed with a similar motivation – merging processing and memory into a homogeneous substrate – but as the name indicates, included many non-local connections: "In an abstract sense, the Connection Machine is a universal cellular automaton with an additional mechanism added for non-local communication. In other words, the Connection Machine hardware hides the details." We are primarily concerned with exposing the details, so that the programmer can decide on resource trade-offs dynamically. However, the implementation of Lisp on the Connection Machine [26] introduces concepts such as xectors which are likely to be useful in the implementation of functional programming languages in our architecture.

One key element of our approach that is not present in any of these models is that of formal conformance to physics:

- classical CAs are an “overshoot” – imposing too many constraints between space and time above those of physics;
- gate arrays have become non-local and are trending further away from local interactions;
- practical parallel languages accept the architecture of commercial computers and simply make the best of it in software; and
- the Connection Machine allows non-local communication by hiding physical details.

Also, at least as important as this is the fact that our model operates precisely without a global clock, while the four models above do not. This decreases power requirements and heat dissipation, while increasing overall speed.

5 Future work

The primary disadvantage to practical fabrication and use of ALA in their present form is the need to simultaneously initialize all cells with the configuration data before useful computation can be performed. We are currently investigating various approaches to solving this problem, such as a protocol for loading the data in to uninitialized space from the edges, by specifying a forwarding direction after each cell is configured and then propagating a final start signal when initialization is finished and computing can begin. We are also developing a hierarchical, module-based design environment for computing this configuration data on a traditional PC.

6 Conclusion

We have presented a new model which merges lattice gases, Petri nets, charge-domain logic, and reconfigurable logic. This model represents a simple strategy for asynchronous logic design: making the operations asynchronous at the bit level, and not just at the level of pipelines and functional blocks. It is also a potentially faster, more efficient, and lower-power alternative to traditional FPGAs, or to general-purpose computers for highly parallelizable tasks.

References

[1] Ronen, R., Mendelson, A., Lai, K., Lu, S.L., Pollack, F., Shen, J.P.: Coming challenges in microarchitecture and architecture. Proceedings of the IEEE **89**(3) (2001) 325–340

[2] Werner, T., Akella, V.: Asynchronous processor survey. Computer **30**(11) (1997) 67–76

[3] Geer, D.: Is it time for clockless chips? Computer **38**(3) (March 2005) 18–21

[4] Kurzweil, R.: The Singularity Is Near : When Humans Transcend Biology. Viking Adult (September 2005)

[5] von Neumann, J.: Theory of Self-Reproducing Automata. University of Illinois Press (1966)

[6] Fredkin, E., Toffoli, T.: Conservative logic. International Journal of Theoretical Physics **21**(3) (April 1982) 219–253

[7] Denker, J.S.: A review of adiabatic computing. In: Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium. (1994) 94–97

[8] Morita, K., Imai, K.: Logical universality and self-reproduction in reversible cellular automata. In: ICES '96: Proceedings of the First International Conference on Evolvable Systems, London, UK, Springer-Verlag (1996) 152–166

[9] Lee, J., Peper, F., Adachi, S., Morita, K., Mashiko, S.: Reversible computation in asynchronous cellular automata. In: UMC '02: Proceedings of the Third International Conference on Unconventional Models of Computation, London, UK, Springer-Verlag (2002) 220–229

[10] Toffoli, T., Capobianco, S., Mentrasti, P.: When – and how – can a cellular automaton be rewritten as a lattice gas? (September 2007)

[11] Petri, C.A.: Nets, time and space. Theoretical Computer Science **153**(1-2) (January 1996) 3–48

[12] Hénon, M.: On the relation between lattice gases and cellular automata. In: Discrete Kinetic Theory, Lattice Gas Dynamics, and Foundations of Hydrodynamics. World Scientific (1989) 160–161

[13] Frisch, U., d'Humières, D., Hasslacher, B., Lallemand, P., Pomeau, Y., Rivet, J.P.: Lattice gas hydrodynamics in two and three dimensions. In: Lattice-Gas Methods for Partial Differential Equations. Addison-Wesley (1990) 77–135

[14] Murata, T.: State equation, controllability, and maximal matchings of petri nets. IEEE Transactions on Automatic Control **22**(3) (1977) 412–416

[15] Sangster, F.L.J., Teer, K.: Bucket-brigade electronics: new possibilites for delay, time-axis conversion, and scanning. IEEE Journal of Solid-State Circuits **4**(3) (June 1969) 131–136

[16] von Neumann, J.: First draft of a report on the EDVAC. IEEE Annals of the History of Computing **15**(4) (1993) 27–75

[17] Smith, A.R.: Cellular Automata Theory. PhD thesis, Stanford University (1970)

[18] Banks, E.R.: Cellular Automata. Technical Report AIM-198, MIT (June 1970)

[19] Cook, M.: Universality in elementary cellular automata. Complex Systems **15**(1) (2004)

[20] Shoup, R.G.: Programmable Cellular Logic Arrays. PhD thesis, Carnegie Mellon University (1970)

[21] Minnick, R.C.: Cutpoint cellular logic. IEEE Transactions on Electronic Computers **EC-13**(6) (December 1964) 685–698

[22] Roscoe, A.W., Hoare, C.A.R.: The laws of Occam programming. Theoretical Computer Science **60**(2) (September 1988) 177–229

[23] Armstrong, J., Virding, R., Wikström, C., Williams, M.: Concurrent Programming in Erlang, Second Edition. Prentice-Hall (1996)

[24] Steele, G.L., Allen, E., Chase, D., Luchangco, V., Maessen, J.W., Ryu, S., Tobin-Hochstadt, S.: The Fortress Language Specification. Technical report, Sun Microsystems (March 2007)

[25] Hillis, W.D.: The Connection Machine. MIT Press, Cambridge, MA (1985)

[26] Steele, G.L., Hillis, W.D.: Connection Machine Lisp: fine-grained parallel symbolic processing. ACM Press (1986)

.

A 1D cellular automaton that moves particles until regular spatial placement

Luidnel Maignan[1] and Frédéric Gruau[2,1,3,4]

[1] INRIA Futurs Saclay - 4, rue Jacques Monod, 91893 ORSAY Cedex, France,
[2] LRI - Université Paris-Sud 11, bâtiment 490, 91405 Orsay Cedex, France
[3] LIRMM - 31 rue Ada, 34000 Montpellier, France
[4] UWE, Frenchay Campus, Coldharbour Lane, Bristol BS16 1QY, United Kingdom
[**]

Abstract. We consider a finite Cellular Automaton (CA) with particles where in each site, a bit encodes presence or absence of a particle. Starting from an arbitrary initial configuration, our goal is to move the particles between neighbor sites until each particle is placed at the same distance from its neighbors. This problem, called "regular placement" problem, is the CA-equivalent of load-balancing in parallel computing. We present a CA rule that solves this problem in the 1D case, and is convergent, i.e, once the regular placement is achieved, the configuration does not change anymore. The rule is inspired from the Lloyd algorithm, computing a centroidal Voronoi tesselation. The dynamic of the rule is described using self-explanatory spatio-temporal diagram. They show that particles move upon reception of signals carrying the energy of the system. Each signal bounces or pass through particle, until it eventually reaches the border and vanishes. When signals have all vanished, particles are regularly placed.

1 Introduction

A *cellular automaton* (CA) is a discrete space, called *cellular space*. It is defined by a lattice of *sites* having their own *state*, and evolving in discrete time according to a *rule*, which determines the next state of each site depending on its current states along with the current states of its neighbors. A *configuration* is the set of every states at a given time [14, 3].

This article tackles the following problem: given an initial configuration with an arbitrarily finite set of sites holding a particle each, find a rule that moves the particles continuously until convergence in a configuration where they are uniformly placed in the cellular space.

[**] We acknowledge support from EPRC grant EP/F003811/1 on general purpose spatial computation.

This problem is a discrete version of a more general problem considering the placement of a set of points, which can be robots, telephone antenna, or real points in a 3D or visualization algorithm. The uniformity constraint is well defined by formulating the problem as a particular occurrence of the Least Squares Problem (LSP): find a set of points minimizing the squared distance to nearest point averaged through the considered region [9, 4, 2]. It is proved that LSP optimal solutions correspond to Centroidal Voronoi Tesselations (CVT), which means that the set of points correspond to the centers of mass of the Voronoi regions that they generate [4]. In other words, nearest neighbors tend to be equidistant, and achieve a regular arrangement [12]. Figure 1 [5] gives a clear view of the input and the desired output with a 2D euclidean space, and the relation with the Voronoi tesselation. In a CVT, it is clear that points are uniformly distributed.

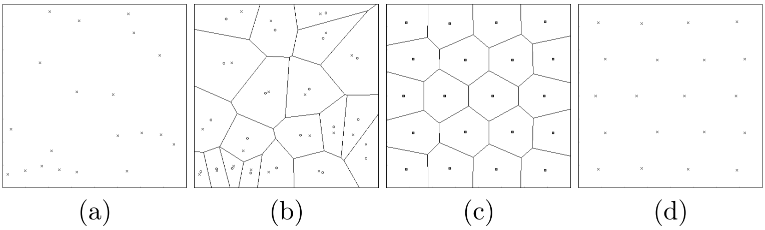


Fig. 1. (a) 20 randomly positioned generators; (b) Voronoi tesselation of these generators, the circles are the centers of mass of Voronoi regions; (c) a centroidal Voronoi tesselation, circle and crosses are merged since generators are centers of mass of theirs Voronoi regions; (d) the generators of the centroidal Voronoi tesselation.

Nevertheless, the classic study of CVT does not consider a given initial set of points to whom the algorithm is stuck, but rather an integer n allowing the algorithm to generate its own initial set of n points. Furthermore, our problem imposes small movements on the points. These constraints also hold for problems like self-deployment of mobile sensors [13], or in the area coverage problem in robotics where the initial set of points correspond to physical objects already positioned.

Our real motivation is the following: solving our full constrained problem in a robust and efficient way is crucial for a model of massive parallelism called blob computing, developed by the authors [6, 5]. In this model, the architecture is a CA and each site can host a thread of computation (with finite state). Threads can create new threads, so it is important to constantly homogenize thread density to avoid congestion. We associate one particle to each thread and

[5] generated from the Christian Raskob's Java applet that can be found at: `http://www.raskob.de/fun/d/CentroidalVoronoi.tgz`

make the thread's state move to follow the particle move. Then, the problem of homogenization of thread density becomes equivalent to the problem considered in this article. More generally, it is a way to implement dynamic load balancing, which is a central problem in parallel computing.

There are two main way of computing CVT: Lloyd's algorithm [9] which is iterative and move each points to the center of mass of its Voronoi region until a convergence condition, and the McQueen method [10], also called the k-mean method, which is also iterative but determine the CVT without computing any Voronoi tesselation. A huge variety of variants from Lloyd and McQueen's method exist [4], and a parallel implementation is introduced in [8].

The CA presented here is a simplified spatial discretization of the Lloyd's algorithm, since the considered problem is a specific case of the CVT. The space is unidimensional and discrete, and the density is constant. These restrictions make the problem look very simple. Let $x_1 \dots x_k$ be the particles, and $p_1 \dots p_n$ the sites. The particle x_i should be placed more or less to sites $p_{(i-\frac{1}{2})\frac{n}{k}}$. However, the problem is not trivial as we bound the memory of the sites independently from the number of sites n or the number of points k. It implies that one cannot use the preceding centralized algorithm that computes the precise target destination for each points. Instead, each site must continuously probe the density in its neighborhood, and move its point towards the region of lower density. Also, finite state prevent to use a unique id for each point.

The same problem can be considered in 2D or 3D, which leads to more difficulty and is an ongoing work. Adamatzky [1], followed by Maniatty and Szymanski [15], already describe how to compute the Voronoi tesselation in a static way on a CA. These works can be extended to allow dynamic computation of the Voronoi tesselation, as the generators (or particles) move. The finite state constraint is achieved by emitting waves from the generators, but this prevents a simple definition of the convergence. Using the same principle of waves emission, dynamic Voronoi tesselation can be computed on excitable media [1, 7]. The determination of the center of mass in a local manner can also be done using vector field [11] or other techniques, which leads to a CVT when coupled with a dynamic Voronoi tesselation algorithm.

Our CA is obtained by coupling two symmetric layers running the same CA rule. We present the rule in the 1D context: in this restricted case, we are able to provide a proof of the correctness. Two versions of the automaton are presented, one that uses integer numbers and, as a consequence, has an infinite number of states, and another one that uses properties of the first one to formulate it with a bounded number of states, namely 7 states per layer. Last we further modify the rule to guarantee that the CA converges.

2 Cellular automaton with integer numbers

In this section, we start by describing the global approach of the algorithm used in our cellular automaton, showing the link with Lloyd's algorithm and its organization in two layers executing the same rule. Then we describe each layer independently, to finish by showing more precisely how the two layers interact to form the CA rule. The CA rule introduced in this section manipulates integers.

2.1 The global approach

The algorithm used in our cellular automaton to solve the given problem is in the same spirit as the Lloyd's algorithm. The former consists in repeating the following stages until the resulting set of points is good enough:

1. Compute the Voronoi regions $\{V_i\}_{i=1}^{k}$ of the current set of points $\{P_i\}_{i=1}^{k}$
2. Compute the new set of point $\{P_i\}_{i=1}^{k}$ as the centers of mass of $\{V_i\}_{i=1}^{k}$

In the 1D case, the Voronoi regions are intervals, allowing to represent them by their boundaries as a set $\{W_j\}_{j=1}^{k+1}$ such that $V_i = [W_i, W_{i+1}]$. Considering that the Voronoi region boundaries, called walls for brevity, are bisectors which are middle points in the 1D case, and that the center of mass of an interval with uniform density is its middle point, both concepts leads to the same computation:

1. $\{W_j\}_{j=1}^{k+1} \leftarrow Middles(\{P_i\}_{i=0}^{k+1}) = \{\frac{1}{2}(P_i + P_{i+1})\}_{i=0}^{k}$
2. $\{P_i\}_{i=1}^{k} \leftarrow Middles(\{W_j\}_{j=1}^{k+1}) = \{\frac{1}{2}(W_j + W_{j+1})\}_{j=1}^{k}$

Our approach consists of using two symmetric layers: one layer encodes the points positions and compute their middles, while the other encodes the walls positions and compute their middles. Also, the points positions are updated in the first layer according to the middles computed by the second layer and the walls positions are updated in the second layer according to the middles computed by the first layer.

To express the symmetry, we introduce the term *source* that refers to the points regarding to the first layer, or the walls regarding to the second layer. It is clear that each layer runs the same rule, that we call "layer rule". This rule receive the middles of the other layer as destinations for its sources, and update its own middles according to the current positions of its source.

This layer rule behavior is obtained by computing for each site its distance to its nearest source. From these distances, the middles are obtained by detecting local non-strict maximum sites, called *summits*. Because of local propagation, this computation takes several transitions to complete, but instead of waiting for an exact set of middles, we update the two layers at each transition. It saves synchronization mechanisms and allows the points and the walls to be updated as soon as possible, leading to continuous displacement of both sets of middles/summits.

2.2 Description of the layer rule

We begin by describing the layer rule. For the moment, we only describe the computation of the distances to the nearest source and consider that the source position is provided by a Boolean which is true in sites holding a source.

The distance to the nearest source is the length of the shortest path to a source. A classical way to compute these distances is to consider, for each site, the length of paths starting from each neighbors, take the shortest one, and put the additional step to this path to obtain the shortest path from the considered site to a source. In terms of distance, it gives the following well-known rule:

$$c_{t+1}(i) = \begin{cases} 0 & \text{if the site } i \text{ contains a source} \\ 1 + \min\{c_t(j)\}_{j \in neighbors(i)} & \text{otherwise.} \end{cases} \tag{1}$$

Let us study this rule under different conditions. Firstly, we consider the classical case where the sources do not move. This leads to the execution shown in Fig. 2:

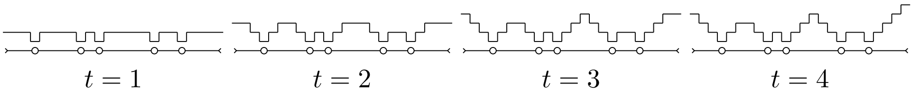


Fig. 2. Evolution of the rule (1) through 4 instants. The source's positions are indicated by the balls under the distances whose values are represented by different levels. The configuration has 0 in every sites at time 0. The last configuration is $[3, 2, 1, 0, 1, 2, 2, 1, 0, 1, 0, 1, 2, 3, 2, 1, 0, 1, 1, 0, 1, 2, 3, 4]$.

We consider the following definition of the summits:

$$summit(c)(x) = c(x) \geq \max(c(x-1), c(x+1)) \wedge c(x) > 0, \tag{2}$$

which only says that they are local non-strict maximum having a non-null value. The summits span the whole intervals between the sources at time 1, and are shrunk into intervals of size 1 or 2 during the execution. If we consider that a summit of size 2 has for position the center of its interval, we obtain that the summit's positions correspond to the middles as required.

Let us consider now the case of a displacement occurring on a source. First, we take two sources, execute the rule (1) until convergence, and then move the right source one site to the right. Figure 3(a) shows the execution of rule (1) from the obtained configuration. At time $t = 1$, the displacement is visible where the right source (right ball) is one unit on the right of the null distance site. For brevity, we call these null distance sites *valley*, because of their local minimum status.

Considering the definition of summits given in Eq. 2, that execution leads to the creation of a spurious summit. This is because the previous site of the source has two neighboring sites with value 1. So it jumps to the value 2 immediately

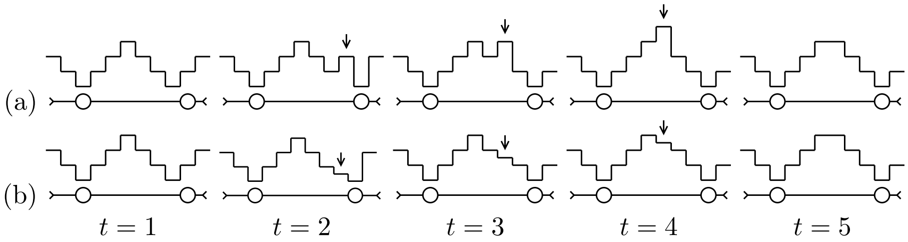


Fig. 3. (a) Execution of the rule (1) after a source displacement. The down arrow indicate the spurious summit position. (b) Execution of the rule (3). The down arrow indicate the $-\epsilon$ positions

after loosing its source. In order to conserve the equivalences between summits and middles, we need to modify the rule to prevent those spurious summits to be created. The rule (3) is like (1) with the additional line: $T_I(c,s)(i) = 1-\epsilon$ if $\neg s(i) \wedge c(i) = 0$, which detects the sites that have just lost a source, and gives them the value $1-\epsilon$. This value is strictly between the values of its two neighbors at the next instant (0 for the neighbors receiving the source, 1 for the other) which prevent the detection of a summit. Figure 3(b) shows the execution of the rule (3).

$$T_I(c,s)(i) = \begin{cases} 0 & \text{if } s(i), \\ 1-\epsilon & \text{if } \neg s(i) \wedge c(i) = 0, \\ 1+\min(c(i-1), c(i+1)) & \text{otherwise.} \end{cases} \tag{3}$$

The rule (3), named T_I, exhibits every needed properties and corresponds to our layer rule. It uses integer numbers, leading to an unbounded number of states. We will consider later an other versions T_F that directly inherits from this rule and has a finite number of state. We also delay the precise description of the dynamics of the rule (3) for the moment, to concentrate now on the way that rule is combined with itself when considering the two layers described in Sect. 2.1.

2.3 The CA rule: coupling two layer rules

We have described the layer rule which compute the middles from the sources for a layer. To obtain the CA rule, we need to combine two layers and make each uses the summits of the other to update its sources position. Let us see now how the sources are represented.

In the layer rule, we have considered that it is possible to have a Boolean indicating the source presence in each site. A first natural idea is to encode this Boolean directly in the state of each site. The updating of this Boolean should then move the source to the middles smoothly (less than one site per instant). However, since the behavior of the layer rule makes the summits move smoothly, the sources will do more than follow the summits: they will be equal to the summits of previous instant. So we simplify the CA rule by directly providing the summits of each layer as the sources of the other.

So the two layers are represented by two integer numbers p and w on each site. p is considered to be the distance to the nearest point, and w to be the distance to the nearest wall. In the rest of the article, $p_t(i)$ denote the value of p in the site i at time t, and $w_t(i)$ denote the value of w in the site i at time t. The two layers evolutions are coupled in the following single CA rule where $T = T_I$ (and also T_F and T_C in the following sections):

$$\begin{cases} p_{t+1} = T(p_t, summit(w_t)), \\ w_{t+1} = T(w_t, summit(p_t)), \end{cases} \tag{4}$$

The final concern is the initial configuration. The layer p associates a valley to sites that correspond to particles, and the distance 1 to every other sites. Thus, the summits of the layer p are large intervals and will be shrunk as showed in Fig. 2. The layer w has the value 1 where p has the value 0, and vice-versa, thus creating an initial correspondence between the valleys of each layer with the summits of the other.

$$\begin{cases} p_0(i) = 0 \text{ if } i \in \{P_j\}_{j=1}^k, 1 \text{ otherwise}, \\ w_0(i) = 1 \text{ if } i \in \{P_j\}_{j=1}^k, 0 \text{ otherwise}. \end{cases}$$

The behavior on the border of the cellular space is defined by the reflection boundary condition, which means that if the space size is n, $c_t(-1) = c_t(1)$ and $c_t(n) = c_t(n-2)$ for $c = p$ and $c = w$. The resulting cellular automaton, coupling the two layer, exhibits really interesting behavior, some happening in each layer separately, and others being the result of the coupling. Those behaviors reflects how the cellular automaton works, and are explained in the following sections.

3 Dynamics of the cellular automaton

In this section, we explain the behavior of our cellular automaton. We identify signals traveling between particles and walls and causing their displacements in the cellular space. From the interaction between the particles, the walls and the signals, we abstract the behavior of our two-layered cellular automaton into a consistent spatio-temporal diagram showing how the cellular automaton evolves as a whole. In the following sub-sections, we consider maximum intervals where the value on a layer is constant. They include only valleys and summits, enabling to detect summits locally as explained in Sect. 2.2. The slopes are the remaining parts between the valleys and the summits. Figure 4 illustrates these definitions.

3.1 Nature of the signals

It is possible to define signals firstly by identifying a preserved information, and secondly by showing the way they move in the cellular space. In those two respects, Fig. 3 already gives a lot of information on the signals characterizing the

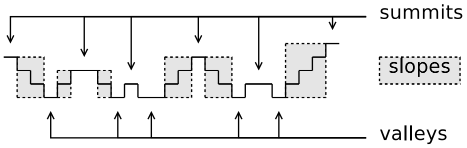


Fig. 4. Different parts of a layer of the CA. Slopes are always strict. They can have a null size, but we consider that there is always a slope between a valley and a summit

rule (3). In fact, the spurious summit on row (a), as well as the special pattern created by the $-\epsilon$ on row (b) are conserved and move from the valley, that creates that, all the way to the summit that absorbs them. Figure 5(a) illustrates that fact by showing explicitly the transformation occurring on the slopes: they are just translated to obtain the next configuration. The vector of translation[6] is $(1, 1)$ for increasing slopes, and $(-1, 1)$ decreasing slopes.

Figure 3 also shows that creation of a signal is done by a displacement of a valley and its absorption results in the displacement of a summit. From interactions of w and p layers (Eq. 4), the summit's displacement in one layer causes the valley's displacement in the other layer, since the summits of each layer become the valleys of the other. Thus, signals absorbed in one layer are transferred to the other layer. Figure 5(b) gives an abstract view of the translation occurring in the whole cellular automaton, along with the transfers from summits to valleys of the cellular automaton. It can be summarized by a virtual network wall $\rightleftharpoons$ point $\rightleftharpoons$ ... wall... $\rightleftharpoons$ point $\rightleftharpoons$ wall.

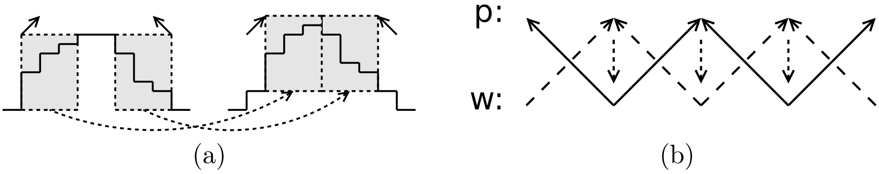


Fig. 5. (a) on the left, a configuration; on the right, the configuration after transformation. The same vectors are represented on both side to highlight the occurring translation. (b) The underlying movement of signals in the whole system: the set of all translation, plus the transfer of signal from the displacement of a summit to the displacement of the correspond valley (down arrows in dotted lines)

[6] A translation (dx, dy) has to be interpreted as $c_t(x + dx) = c_{t-1}(x) + dy$

3.2 Signals and energy

Figure 5(a) shows that patterns move in this virtual network. Since the pattern move in x-axis but also in the y-axis, the conserved information is the differential in the pattern. So the signals are located between two sites i and $i+1$, and are related to $c_t(i+1) - c_t(i)$.

Figure 3(1) shows a configuration obtained with two sources fixed until convergence. So no signals should be in it. This remark complete our definition of the signals: it is the difference between the current differential and the desired one. The desired differential is 1, -1 or 0, depending on whether the two sites are in a positive slope, a negative or a two-sites non-strict maximum or minimum. Care should be taken that the minimums are defined by the source presence instead of the valley, allowing to take into account the signals during their transfer from one layer to the other. Figure 6 shows schematically the correlation between the displacement of a valley, the generated signal, and the displacement of the corresponding summit.

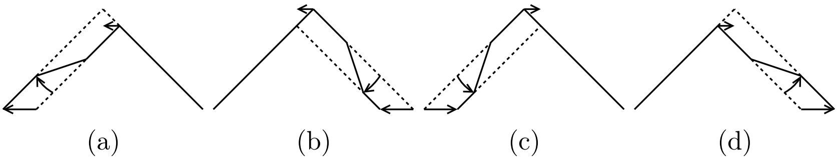


Fig. 6. Effect of a valley displacement to the left(a,b) and to the right(c,d) on positive(a,c) and negative(b,d) slopes, and the resulting summit displacement. The signs of the signals are represented by the clockwiseness of the arcs showing the slopes modification.

By studying the sign of signals created by a valley's displacement, it is possible to see that the signals is positive if the displacement is rightwards, and negative if it is leftwards. Also, a valley's displacement towards a direction creates a displacement of a summit in the same direction. So the sign of signals fully corresponds to directions.

Let us now consider all the signals together. We define the energy of the system has the sum of the absolute values of signals. The preceding reasoning shows that energy does not increase. In fact, it decrease because signals reaching the border of the CA vanish without creating additional displacement. Also, signals of opposite value do not create additional displacement when they meet on a summit, since they should create movement in opposite direction. So they annihilate each other and the energy also decreases in this case. When the energy becomes zero, all slopes are exactly 1 or -1, and the distances between any valley (resp. summits) and its two neighbor summits (resp. valleys) are the same, thus achieving a regular placement.

Figure 7 exhibits a whole execution by showing only the valleys of both layers (points and walls) and the signals traveling in the space and interacting with the valleys. The initial configuration is a compact set of points placed slightly to the left of the center of the space. The final configuration (bottom of the second column) shows a regularly placed set of points.

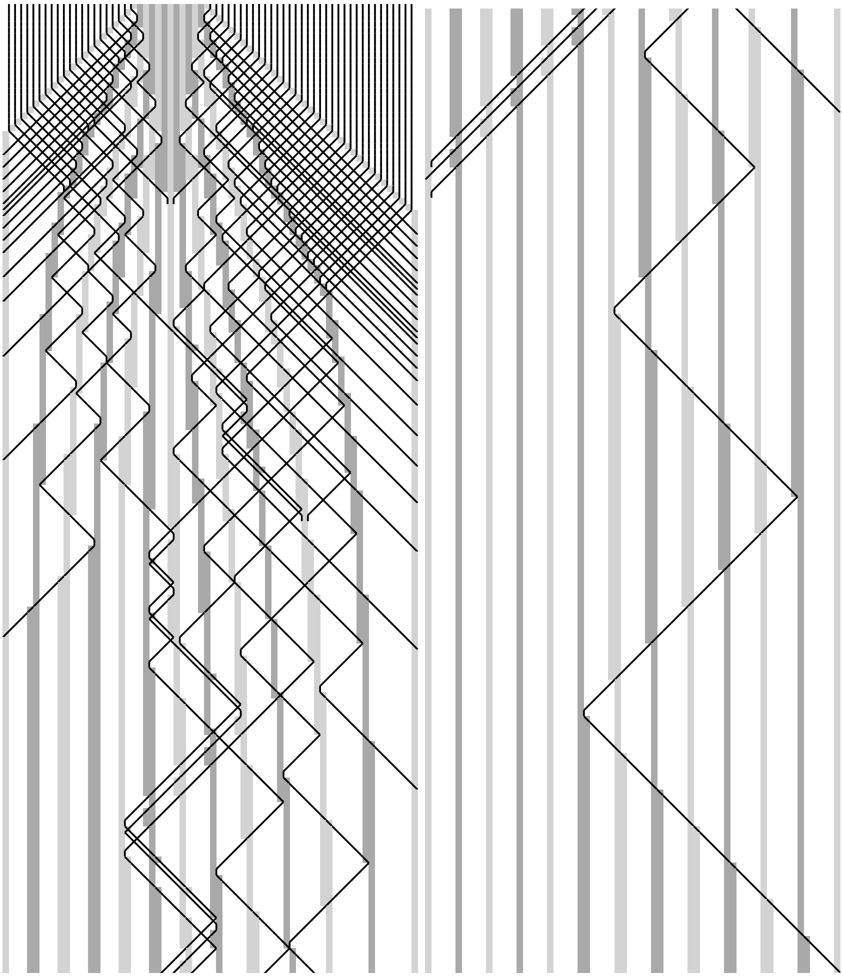


Fig. 7. Spatio-temporal diagram representing the points (dark gray), the walls (light gray) and the signals traveling through the space (black)

4 Cellular automaton with finite state

The rule (3) uses integer numbers. The number of used states is a factor of the size of the space. We show now that its possible to use only a fixed number of states. The thing to notice is that most of the time, the difference between the values of two consecutive sites is small. If this difference is bounded, only a fixed number of local configuration is needed.

To obtain the finite state cellular automaton, we modify the rule to bound the difference of values of two neighbor sites, without breaking the dynamics of the cellular automaton. Then we bound the number of state by storing the value of the state modulo a certain number. In the rest of this section, the rule is written with $\epsilon = 0.5$ and every values are doubled in order to have only integers. Using this convention, the rule (3) becomes:

$$T_I(c,s)(i) = \begin{cases} 0 \text{ if } s(i), \\ 1 \text{ if } \neg s(i) \wedge c(i) = 0, \\ 2 + \min(c(i-1), c(i+1)) \text{ otherwise.} \end{cases}$$

4.1 Bounding the differences

We already showed that every differences are just translated in the slope, and are created in the valleys when they move. It means that we only have to take care of what happens in the valleys to bound every differences in the system. When a site looses its sources, the difference created is always $0.5 * 2 = 1$. The difficult case is the site winning a source. Figure 8 shows that if a source moves in the same direction repeatedly, arbitrary large differences are created.

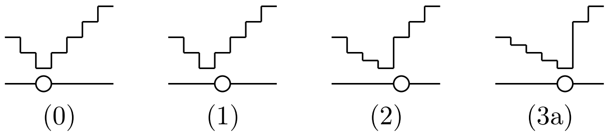


Fig. 8. Effects of a source moving two times in the same direction

Those large differences are due to the direct assignment to 0 of a site having a source. They disappear if we allow a site to smoothly decrease of α at a time, with $\alpha = 1$ or 2. So rule part: $T_I(c,s)(i) = 0$ if $s(i)$, of the rule (3) becomes: $T_F(c,s)(i) = max(0, c(i) - \alpha)$ if $s(i)$ of the rule (5).

$$T_F(c,s)(i) = \begin{cases} max(0, c(i) - \alpha) & \text{if } s(i), \\ 1 & \text{if } \neg s(i) \wedge c(i) = 0, \\ 2 + \min(c(i-1), c(i+1)) & \text{otherwise.} \end{cases} \quad (5)$$

However, as Fig. 9(3b) shows, rule (5) can create a valley with a non-null distance. Therefore, we need to modify this rule so that a valley keeps its value

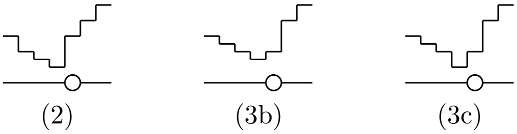


Fig. 9. Situation after (2) using a modified rule: (3b) with smooth decrease, and (3c) with smooth decrease and preservation condition

as long as the smooth decrease is not finished. The condition (6) is true when a valley has to be preserved. The rule (7) implement this valley preservation.

Figure 10 shows that the run with $\alpha = 2$ is almost the same as the integer CA. A close inspection reveals only a dozen of spots where the delay implied by the rule (7) are represented by a discontinuity in the signal.

$$cond_F(c,s)(i) = c(i) = 0 \land \max(c(i-1), c(i+1)) > \alpha \tag{6}$$

$$T'_F(c,s)(i) = \begin{cases} c(i) & \text{if } cond_F(c,s)(i), \\ T_F(c,s)(i) & \text{otherwise.} \end{cases} \tag{7}$$

4.2 Bounding the number of states

The maximum difference is $k = 4$, when $\alpha = 2$ and $k = 3$ when $\alpha = 1$. The rule (7) can be left as is if one store the state (mod $2k+1$), and add a preliminary stage where $v = 3k + 1 - c(i)$:

$$\begin{aligned} c'(i) &\leftarrow k, \\ c'(i-1) &\leftarrow (c(i-1) + v) \mod (2k+1), \\ c'(i+1) &\leftarrow (c(i+1) + v) \mod (2k+1). \end{aligned}$$

The effect of this stage is to translate the state values in order to have a correct order relationship between them. Of course, the result of the rule has to be translated back. Those translations do not affect the behavior of the cellular automaton since every notion of summits, signals and so on, are build on top of the differential, which is conserved thanks to the translation process.

The number of state is $2k + 1$. For $\alpha = 1$, $k = 3$ and the number of state for w is 7 and for p also 7. For $\alpha = 2$, $k = 4$ and we need 9 states, but it is quicker.

5 Convergence with probabilistic cellular automaton

The analysis done on energy shows that is decreases. Because it is a positive integer, it converges, but the limit could be non zero. This does happen in some very specific case, as shown in Fig. 11(a).

Depending on whether a summit spans one or two cells, a signal arriving at a summit can go trough it or bonce on it. This can lead to cycles when a

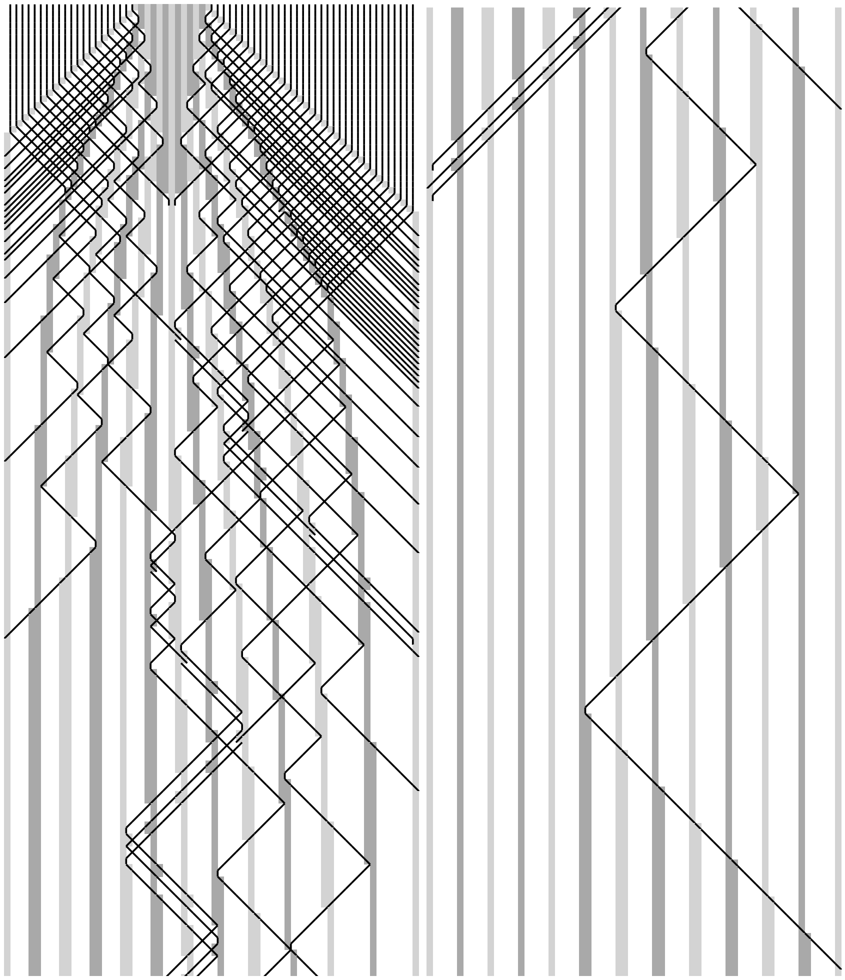


Fig. 10. Spatio-temporal diagram with the same initial configuration as previously, but with the finite state rule

positive signal followed by a negative signal alternatively bounce between two points/walls as show in Fig. 11(a).

Being of opposite signs, those two signals would annihilate each other, if they could meet. To provoke this meeting, we non-deterministically slow down the bouncing of negative signals going rightwards, so that the other positive signal can eventually catch up with the negative.

It suffices to replace $cond_F$ by $cond_F \vee (cond_C \wedge rand(0,1) < p)$ in rule (7), where $cond_C$ defined in (8) is true when the creation of the valley, (corresponding to the bouncing) can be delayed and $rand(0,1) < p$ is true with probability p. Figure 11(b) shows the results.

$$cond_C(c,s)(i) = c(i) \leq 1 \wedge s(i) \wedge s(i+1) \tag{8}$$

6 Conclusion

In this paper, we have presented and explained a 1D CA rule running on two layers with 7 states, and show that it uniformizes a set of points. As shown in the space time diagrams, the rule converges to a final configuration, characterized by an integer d such that any two consecutive points lies at distance d' where $|d - d'| \leq 1$. We analyze the mechanism of the rule to explain this behavior. Our future work will simplify the rule, report a more formal proof of convergence, study the time performance, and generalize to 2D CA.

References

[1] A. Adamatzky. Voronoi-like partition of lattice in cellular automata. *Mathematical and Computer Modelling*, 23:51–66(16), February 1996.

[2] F. Aurenhammer. Voronoi diagramsa survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

[3] B. Chopard and M. Droz. *Cellular Automata Modeling of Physical Systems.* Cambridge University Press, 1998.

[4] Q. Du, V. Faber, and M. Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Rev.*, 41(4):637–676, 1999.

[5] F. Gruau, C. Eisenbeis, and L. Maignan. The foundation of self-developing blob machines for spatial computing. *Physica D*, 2008.

[6] F. Gruau, Y. Lhuillier, P. Reitz, and O. Temam. Blob computing. In *Computing Frontiers 2004 ACM SIGMicro.*, 2004.

[7] J. Jones and M. Saeed. Image enhancement - an emergent pattern formation approach via decentralised multi-agent systems. *Multi-Agent and Grid Systems*, 3(4):105–140, 2007.

[8] L. Ju, Q. Du, and M. Gunzburger. Probabilistic methods for centroidal voronoi tessellations and their parallel implementations. *Parallel Comput.*, 28(10):1477–1500, 2002.

[9] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[10] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[11] L. Maignan. Uniformisation de la rèpartition spatiale d'un ensemble de points par diagramme de voronoi: Implémentation sur automate cellulaire, 2007.

[12] D. J. Newman. The hexagon theorem. *IEEE Transactions on Information Theory*, 28(2):137–138, 1982.

[13] S. Poduri and G. Sukhatme. Constrained coverage for mobile sensor networks, 2004.

[14] T. Toffoli and N. Margolus. *Cellular Automata Machines, A New Environment for Modeling.* The MIT Press, Reading, Massachusetts, 1987.

[15] W. A. Maniatty and B. K. Szymanski. Fine-grain discrete voronoi diagram algorithms. *Mathematical and Computer Modelling*, 26:71–78(8), August 1997.

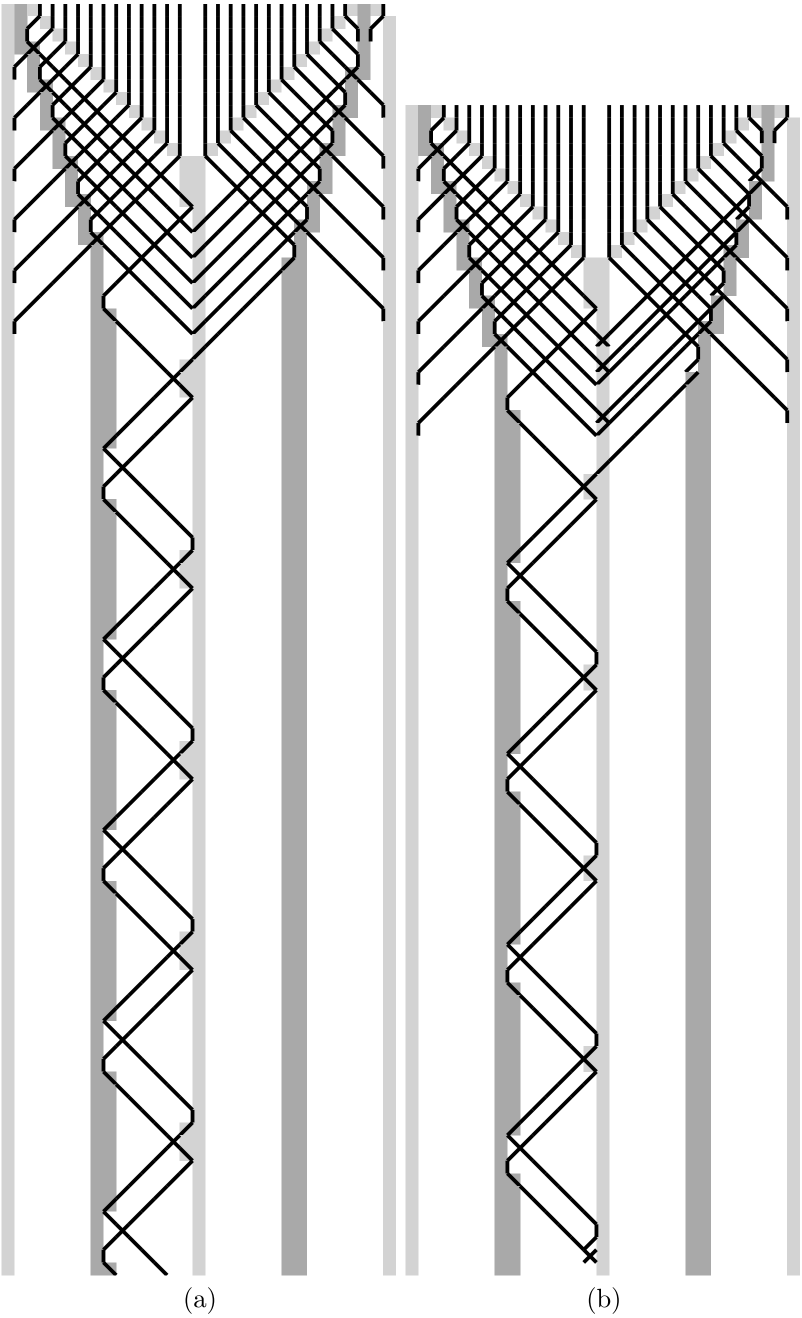


Fig. 11. (a) Spatio-temporal diagram of an initial situation with two particles producing a limit cycle. (b) With the probabilistic rule, the signal is sometimes retained. When it meets its opposing signals, it annihilates with it.

.

An abstract collision system

Takahiro Ito[1], Shuichi Inokuchi[2], and Yoshihiro Mizoguchi[2]

[1] Graduate School of Mathematics, Kyushu University, Japan,
[2] Faculty of Mathematics, Kyushu University, Japan,
t-ito@math.kyushu-u.ac.jp

Abstract. We introduce a new algebraic transition system called an abstract collision system (ACS). We investigate its properties of computations. An abstract collision system is a kind of extension of a billiard ball system. Further, it is considered as an extension of not only a cellular automaton but also a chemical reaction system. Defining collision sets, we formalise ACS as a mathematical algebraic system. Next, we reformulate a billiard ball system using ACS. We also investigate the simulations of one-dimensional cellular automata and cyclic tag systems using ACS. Since simulations are defined by concrete mathematical functions, we can estimate simulation steps by formulae of system parameters.

1 Introduction

Recently, there are many investigations about new computing frameworks which considered as the replacement of current electric computer devices and digital computers. One of main frameworks is the collision-based computing [1] including cellular automata and reaction-diffusion systems. We introduce an algebraic discrete model to describe these kind of models formally and investigate and compare them using theoretical formulations.

When we consider a collision-based computing, a transition is caused by movings(changing) and collisions (reactions). A moving (changing) is a transition of a single object. A collision (reaction) is caused by a set of objects. Conway introduced 'The Game of Life' using two-dimensional cellular automaton [2]. In 'The Game of Life', some patterns in cells called 'glider' is objects. Their collisions are brought by transitions of the cellular automaton. He showed it can simulate any logical operations using 'glider's. Wolfram and Cook [5, 3] found 'glider' patterns which act collisions in the one-dimensional elementally cellular automata CA110. Cook introduced an cyclic tag system (CTS) as a Turing universal system and proved that CTS is simulated by CA110. Recently, Morita [4] introduced a *reversible* one-dimensional cellular automata simulating CTS. In this model, an object is represented by a state itself.

In this paper, we introduce an abstract collision system(ACS). It is a kind of extension of a billiard ball system. Since it is defined as an abstract system, it seems to be considered as an extension of an cellular automaton and chemical

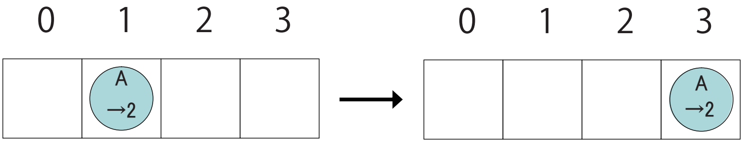


Fig. 1. Moving

reaction systems. First, we show an example of a one-dimensional billiard ball system as ACS. A ball has a velocity, a label and a position. We consider discrete time transitions. A ball moves to left or right according its velocity within an unit time. Let $(2, A, 1)$ be a ball with the velocity 2, the label 'A' and the position 1. At the next step, the ball becomes $(2, A, 3)$ (cf. Fig. 1).

That is the velocity and the label are same and only the position is changed to 3. Some balls may crash in some unit time and exchange their velocities. In our paper, we do not describe a crash using positions and velocities. We define a set of balls which cause collisions and assign the result of the collisions. For example, a collision set is $\{(2, A, 1), (-1, B, 2)\}$. We define the result of the collision by a set $\{(2, B, 3), (-1, A, 1)\}$ and write it as $f(\{(2, A, 1), (-1, B, 2)\}) = \{(2, B, 3), (-1, A, 1)\}$ (cf. Fig. 2).

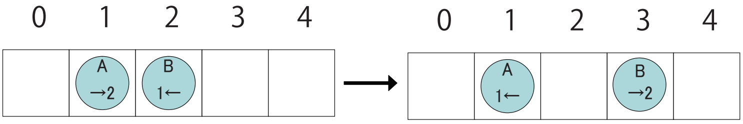


Fig. 2. Collision

For a collision sets $\mathcal{C}$, we prepare a local function $f : \mathcal{C} \to 2^S$ where S is a set of balls and 2^S is the set of all subsets of S. An abstract collision system is defined by a triple $(S, \mathcal{C}, f)$ where S is a set of balls, $\mathcal{C}$ a set of collisions, and $f : \mathcal{C} \to 2^S$.

Let $V = \{-1, 2\}$, $S = \{(u, A, x) \; | x \in Z, \, u \in V\} \cup \{(v, B, y) \; | y \in Z, \, v \in V\}$, and $C = \{ \; \{(2, A, 1), \; (-1, B, 2)\}, \; \{(2, A, 2), \; (-1, B, 3)\}, \; \{(2, A, 1), \; (-1, B, 3)\}, \{(2, A, 1), \; (2, A, 2), \; (-1, B, 3)\} \; \} \cup \{\{s\} | s \in S\}$. We define a function $f : \mathcal{C} \to 2^S$ as follows:

c	$f(c)$
$\{(2, A, 1), (-1, B, 2)\}$	$\{(2, B, 3), (-1, A, 1)\}$
$\{(2, A, 2), (-1, B, 3)\}$	$\{(2, B, 4), (-1, A, 2)\}$
$\{(2, A, 1), (-1.B, 3)\}$	$\{(2, B, 3), (-1, A, 2)\}$
$\{(2, A, 1), (2, A, 2), (-1, B, 3)\}$	$\{(2, A, 3), (2, B, 4), (-1, A, 2)\}$
$\{(u, A, x)\}$	$\{(u, A, x + u)\}$
$\{(v, B, y)\}$	$\{(v, B, y + v)\}$

An abstract collision system is defined by $(S, \mathcal{C}, f)$ and an example of state transitions are figured in Fig. 3.

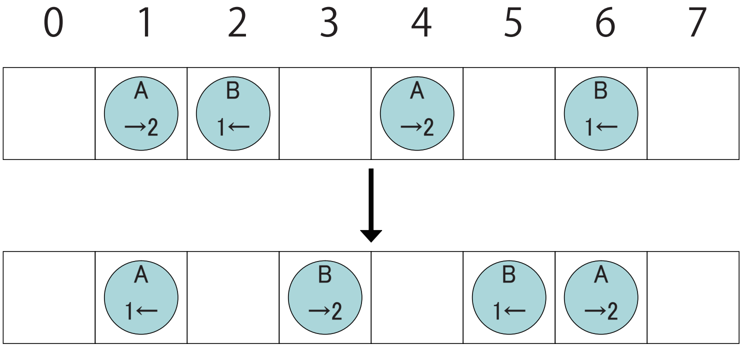


Fig. 3. Transition

We note that $\{(4, A, 2), (-1, B, 6)\}$ does not cause a collision, because it is not in the collision set $\mathcal{C}$ in the example of ACS.

Since an abstract collision system is formalised as an mathematical algebraic system, properties of computations (transitions) are proved rigorously. On the other hand, transitions of an abstract collision systems may have differences from real physical phenomenon. The differences are depend on the definition of a collision set C and a local function $f : C \to 2^S$, so we must choose proper C and f for each real collision phenomenon. We believe we can describe not only one-dimensional billiard ball system but also some general reaction-refusion process and it could be useful to investigate there computation properties.

In Sect. 2, we introduce an abstract collision system using the definition of a collision set. Properties about collision sets are investigated. Further, we reformulate a billiard ball system using our abstract collision system and prove their properties.

In Sect. 3, we simulate a billiard ball system by one-dimensional cellular automaton using our abstract collision system. When a transition of a billiard ball system is simulated by only one transition of a cellular automaton, the relationship between the number of state in a cellular automaton and the number of balls in a billiard system are investigated.

In Sect. 4, we simulate a cyclic tag system using our abstract collision system. The essential idea is similar to the results by Cook [3] and Morita [4]. Our model is defined by functions and formulae, so we can easily describe simulation steps using formulae. Combining the result of this section and the previous section, a cyclic tag system is simulated by one-dimensional cellular automaton. The number of state requiring to simulate a cyclic tag system is investigated. But the minimum state number to simulate a cyclic tag system has not been taken yet.

2 An abstract collision system

In this section, we define an abstract collision system.

Definition 2.1 (Set of collisions). *Let S be a non-empty set. A set $\mathcal{C} \subseteq 2^S$ is called a set of collisions on S iff it satisfies the following conditions:*

(SC1) $s \in S \Rightarrow \{s\} \in \mathcal{C}$.
(SC2) $X_1, X_2 \in \mathcal{C}, X_1 \cap X_2 \neq \phi \Rightarrow X_1 \cup X_2 \in \mathcal{C}$.
(SC3) $A \in 2^S, p \in A \Rightarrow [p]_{\mathcal{C}}^{A} := \cup\{X \mid X \in \mathcal{C}, p \in X, X \subseteq A\} \in \mathcal{C}$.

We note that the condition (SC3) can be omitted if $\mathcal{C}$ is a finite set.

Proposition 2.1. *Let $\mathcal{C}$ be a set of collisions on S. For any $A \in 2^S$ and $p, q \in A$, we have the following:*

(1) $[p]_{\mathcal{C}}^{A} \neq \phi$.
(2) $[p]_{\mathcal{C}}^{A} \cap [q]_{\mathcal{C}}^{A} \neq \phi \Rightarrow [p]_{\mathcal{C}}^{A} = [q]_{\mathcal{C}}^{A}$.

Proof. **(1)** Since $\{p\} \in \mathcal{C}$, $p \in \{p\}$ and $\{p\} \subset A$, we have $\{p\} \subset [p]_{\mathcal{C}}^{A}$. Hence $[p]_{\mathcal{C}}^{A} \neq \phi$.

(2) We assume $[p]_{\mathcal{C}}^{A} \cap [q]_{\mathcal{C}}^{A} \neq \phi$. Since $[p]_{\mathcal{C}}^{A}, [q]_{\mathcal{C}}^{A} \in \mathcal{C}$, $p \in [p]_{\mathcal{C}}^{A} \cup [q]_{\mathcal{C}}^{A} \in \mathcal{C}$ and $[p]_{\mathcal{C}}^{A} \cup [q]_{\mathcal{C}}^{A} \subseteq A$, we have $[p]_{\mathcal{C}}^{A} \cup [q]_{\mathcal{C}}^{A} \subseteq [p]_{\mathcal{C}}^{A}$. Hence $[p]_{\mathcal{C}}^{A} = [p]_{\mathcal{C}}^{A} \cup [q]_{\mathcal{C}}^{A}$. Similarly, we have $[q]_{\mathcal{C}}^{A} = [p]_{\mathcal{C}}^{A} \cup [q]_{\mathcal{C}}^{A}$. Hence $[p]_{\mathcal{C}}^{A} = [q]_{\mathcal{C}}^{A}$.

Definition 2.2 (An abstract collision system). *Let S be a non-empty set and $\mathcal{C}$ be a set of collisions on S. Let $f : \mathcal{C} \to 2^S$. We define an abstract collision system ACS by $ACS = (S, \mathcal{C}, f)$. We call the function f and the set 2^S a local transition function and a configuration of ACS, respectively. We define a global transition function $\delta_{ACS} : 2^S \to 2^S$ of ACS by $\delta_{ACS}(A) = \cup \left\{ f([p]_{\mathcal{C}}^{A}) \mid p \in A \right\}$.*

Example 2.1. Let $S = \{A_1, A_2, A_3, B_1, C_1, C_2\}$,
$\mathcal{C} = \{\{A_3, B_1\}, \{A_1\}, \{A_2\}, \{A_3\}, \{B_1\}, \{C_1\}, \{C_2\}\}$
and $f : \mathcal{C} \to 2^S$ be $f(\{A_3, B_1\}) = \{C_1\}$, $f(\{A_1\}) = \{A_2\}$, $f(\{A_2\}) = \{A_3\}$, $f(\{A_3\}) = \{A_3\}$, $f(\{B_1\}) = \{B_1\}$, $f(\{C_1\}) = \{C_2\}$, $f(\{C_2\}) = \{A_1, B_1\}$.

Then we have $\{A_1, B_1\} \mapsto \{A_2, B_1\} \mapsto \{A_3, B_1\} \mapsto \{C_1\} \mapsto \{C_2\} \mapsto \{A_1, B_1\}$ (cf. Fig. 4).

Proposition 2.2. *For a given relation R_S on S, we define $\mathcal{C}[R_S]$ by*

$$\mathcal{C}[R_S] := \cap \left\{ \mathcal{C} \,\middle|\, \begin{array}{c} \mathcal{C} \textit{ is a set of collisions on } S \textit{ such that} \\ (x, y) \in R_S \Rightarrow \{x, y\} \in \mathcal{C} \end{array} \right\}.$$

Then $\mathcal{C}[R_S]$ is a set of collisions on S.

Proof. **(1)** Let $s \in S$ and $\mathcal{C}$ be a set of collisions on S. Then we have $\{s\} \in \mathcal{C}$ by **(SC1)**. Hence $\{s\} \in \mathcal{C}[R_S]$.

(2) Let $X_1, X_2 \in \mathcal{C}[R_S]$ with $X_1 \cap X_2 \neq \phi$. Let $\mathcal{C}$ be a set of collisions on S such that $(x, y) \in R_S \Rightarrow \{x, y\} \in \mathcal{C}$. Since $X_1, X_2 \in \mathcal{C}$, we have $X_1 \cup X_2 \in \mathcal{C}$ by **(SC2)**. Hence $X_1 \cup X_2 \in \mathcal{C}[R_S]$.

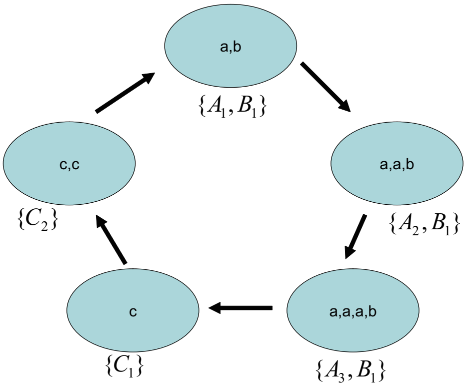


Fig. 4. An example of an abstract collision system.

(3) Let $A \in 2^S$ and $p \in A$. Let $\mathcal{C}$ be a set of collisions on S such that $(x, y) \in R \Rightarrow \{x, y\} \in \mathcal{C}$. Since $\mathcal{C}[R_S] \subseteq \mathcal{C}$, we have $[p]^A_{\mathcal{C}[R_S]} \subseteq [p]^A_{\mathcal{C}} \in \mathcal{C}$ by **(SC3)**. Hence $[p]^A_{\mathcal{C}[R_S]} \in \mathcal{C}[R_S]$.

Next we define a discrete billiard system as a special case of an abstract collision system.

Definition 2.3. *Let L be a finite set, V be a finite subset of $\mathbb{Z}$ and $B = V \times L \times \mathbb{Z}$. We call each element of L a label. We define a relation R_B on the set B by $((v_l, a_l, x_l), (v_r, a_r, x_r)) \in R_B \Leftrightarrow 0 < \frac{x_r - x_l}{v_l - v_r} \leq 1$*

Definition 2.4 (Shift). *For any $X \in 2^B$ and $d \in \mathbb{Z}$, the d-shift of X, which is denoted by $X + d$, is defined by*

$$X + d := \{(v, a, x + d) \mid (v, a, x) \in X\} .$$

We define a relation R_{shift} on B by $(X_1, X_2) \in R_{shift} \Leftrightarrow$ there exists $d \in \mathbb{Z}$ such that $X_2 = X_1 + d$.

Proposition 2.3. *This relation R_{shift} is an equivalence relation.*

Proof. Since

(1) $X = X + 0$,
(2) $X_2 = X_1 + d \Rightarrow X_1 = X_2 + (-d)$ and
(3) $X_2 = X_1 + d_1, X_3 = X_2 + d_2 \Rightarrow X_3 = X_1 + (d_1 + d_2)$,

it is clear that R_{shift} is an equivalence relation.

Definition 2.5. *For the above set B and the relation R_B, consider a set $\cup\{b \mid [b] \in \mathcal{C}[R_B]/R_{shift}\}$ of representative elements.*
It is not determined uniquely. However, we select one of such sets and denote it by $\mathcal{F}[B]$.

Definition 2.6 (A discrete billiard system). *A discrete billiard system is defined by $BS = (B, \mathcal{F}[B], f_{\mathcal{F}[B]})$, where $f_{\mathcal{F}[B]} : \mathcal{F}[B] \to 2^B$ satisfies $f_{\mathcal{F}[B]}(\{(v, a, x)\}) = \{(v, a, x + v)\}$.*

Definition 2.7. *We define $\hat{f}_{BS} : \mathcal{C}[R_B] \to 2^B$ as follows:*

$$\hat{f}_{BS}(X + d) = f_{\mathcal{F}[B]}(X) + d \text{ for } X \in \mathcal{F}[B], d \in \mathbb{Z}. \tag{1}$$

We define a global transition function $\delta_{BS} : 2^B \to 2^B$ of BS by that of an abstract collision system $(B, \mathcal{C}[R_B], \hat{f}_{BS})$.

Remark 2.1. Let L be a set of labels. Let B and $\mathcal{F}[B]$ be those in the above definition. Let $F \subseteq \mathcal{F}[B]$ and $f_F : F \to 2^B$. Then we can define $f_{\mathcal{F}[B]} : \mathcal{F}[B] \to 2^B$ by

(1) $f_{\mathcal{F}[B]}(X) = f_F(X)$ for $X \in F$.
(2) $f_{\mathcal{F}[B]}(X) = \{(v, a, x + v) \mid (v, a, x) \in X\}$ for $X \in \mathcal{F}[B] \setminus F$.

In the following examples, we use F and $f : F \to 2^B$ instead of $\mathcal{F}[B]$ and $f_{\mathcal{F}[B]} : \mathcal{F}[B] \to 2^B$, respectively.

Example 2.2. Let $L = \{A, B\}$, $V = \{-1, 2\}$ and $B = V \times L \times \mathbb{Z}$. Then $F_1 \subseteq \mathcal{F}[B]$, where

$$F_1 = \left\{ \begin{array}{c} \{(2, A, 1), (-1, B, 2)\}, \\ \{(2, A, 1), (-1, B, 3)\}, \\ \{(2, A, 1), (2, A, 2), (-1, B, 3)\} \end{array} \right\}. \tag{2}$$

Let $f_{F_1} : F_1 \to 2^B$ be

$$\begin{aligned} f_{F_1}(\{(2, A, 1), (-1, B, 2)\}) &= \{(2, B, 3), (-1, A, 1)\}, \\ f_{F_1}(\{(2, A, 1), (-1, B, 3)\}) &= \{(2, A, 3), (-1, B, 2)\}, \\ f_{F_1}(\{(2, A, 1), (2, A, 2), (-1, B, 3)\}) &= \{(-1, A, 2), (2, A, 3), (2, B, 4)\}. \end{aligned} \tag{3}$$

Then we have

$$\begin{aligned} &\{(2, A, 1), (-1, B, 2), (2, A, 4), (-1, B, 6)\} \\ \mapsto &\{(-1, A, 1), (2, B, 3), (-1, B, 5), (2, A, 6)\} \end{aligned} \tag{4}$$

(cf. Fig. 3).

Example 2.3. Let $V = \{0, 1\}$, $L = \{\varepsilon, Y, N, T, H, A, R, R'\}$ and $B = V \times L \times \mathbb{Z}$. Then $F_2 \subseteq \mathcal{F}[B]$, where

$$F_2 = \left\{ \begin{array}{ll} \{(1, H, -1), (0, Y, 0)\}, & \{(1, H, -1), (0, N, 0)\}, \\ \{(1, A, -1), (0, Y, 0)\}, & \{(1, A, -1), (0, N, 0)\}, \\ \{(1, A, -1), (0, T, 0)\}, & \{(1, R', -1), (0, Y, 0)\}, \\ \{(1, R', -1), (0, N, 0)\}, & \{(1, R', -1), (0, T, 0)\}, \\ \{(1, R, -1), (0, Y, 0)\}, & \{(1, R, -1), (0, N, 0)\}, \\ \{(1, R, -1), (0, T, 0)\} & \end{array} \right\}. \tag{5}$$

Table 1. The local function $f_{F_2}(\{(1,a_l,-1),(0,a_r,0)\})$.

		a_r		
		Y	N	T
a_l	H	$\{(0,\varepsilon,0),(1,A,0)\}$	$\{(0,\varepsilon,0),(1,R',0)\}$	$----$
	A	$\{(0,Y,0)(1,A,0)\}$	$\{(0,N,0),(1,A,0)\}$	$\{(0,\varepsilon,0),(1,\varepsilon,0)\}$
	R'	$\{(0,Y,0)(1,R',0)\}$	$\{(0,N,0),(1,R',0)\}$	$\{(0,\varepsilon,0),(1,R,0)\}$
	R	$\{(0,\varepsilon,0),(1,R,0)\}$	$\{(0,\varepsilon,0),(1,R',0)\}$	$\{(0,T,0),(1,\varepsilon,0)\}$

Moreover, we define a local transition function $f_{F_2} : F_2 \to 2^B$ by Table 1 below.

We define $f_{\mathcal{F}[B]} : \mathcal{F}[B] \to 2^B$ by

$$f_{\mathcal{F}[B]}(X) = \begin{cases} f_{F_2}(X) & \text{if } X \in F_2, \\ f_{\mathcal{F}[B]}(X') \cup \{(v_0,\varepsilon,x_0+v_0)\} & \text{if } X = X' \cup \{(v_0,\varepsilon,x_0)\}, \\ \{(v,a,x+v) \mid (v,a,x) \in X\} & \text{otherwise.} \end{cases} \tag{6}$$

Transition examples of this discrete billiard system is described the following remark (cf. Fig. 5, Fig. 6 and Fig. 7).

We study about simulations in the following sections. The above example is very important to show that a discrete billiard system simulates a cyclic tag system.

Remark 2.2. If a ball with label H collides with another ball with label Y or N, it produces a ball with label A (Acceptor) or R' (Rejector), respectively (**Fig. 5**). A ball with label A passes through all balls with label Y or N. If it collides a ball with label T, they disappear, that is, They change into balls with ε (**Fig. 6**). On the other hand, a ball with label R' passes through all balls with label Y or N. If it collides a ball with label T, it produces a ball with label R. This ball with label R sweeps out all balls with label Y or N until it encounters a ball with label T (**Fig. 7**).

3 Cellular automata and ACS

In this section, we study about the simulation. First we show that a billiard system with special properties can be simulated by a cellular automaton.

Definition 3.1 (A cellular automaton). *Let Q be a non-empty finite set of states of cells. Let $f : Q^3 \to Q$ be a local transition function. A three-neighbor cellular automaton is defined by $CA = (Q, f)$. A configuration of CA is a mapping $q : \mathbb{Z} \to Q$. The set of all configurations is denoted by $Conf(Q)$. The global transition function $g : Conf(Q) \to Conf(Q)$ of CA is defined as follows: $g(q)(i) = f(q(i-1), q(i), q(i+1))$.*

Definition 3.2. *Let $BS = (B, F[B], f_B)$ be a discrete billiard system, where $B = L \times V \times \mathbb{Z}$. We define a function $pt : 2^B \times \mathbb{Z} \to 2^{V \times L}$ by $pt(A, x_0) = \{(v,a) \mid (v,a,x_0) \in A\}$.*

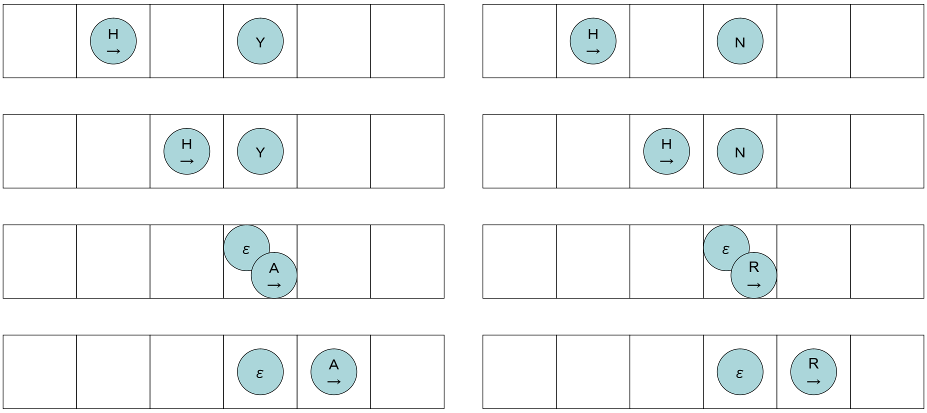


Fig. 5. A ball with label H collides with another ball with label Y or N, then it produces a ball with label A or R, respectively.

Proposition 3.1. *Let L be a non-empty finite set, $V := \{0, +1\}$ $B := V \times L \times \mathbb{Z}$ and BS be a discrete billiard system $(B, \mathcal{F}[B], f_{\mathcal{F}[B]})$. We assume that the function $f_{\mathcal{F}[B]}$ satisfies $f_{\mathcal{F}[B]}(\{(1, b, x-1), (0, a, x)\}) = \{(0, a', x), (1, b', x)\}$, $(a, b, a', b' \in L)$. Then there exists a cellular automaton $CA = (Q, f)$ and a function $\pi : 2^B \to Conf(Q)$ such that $g \circ \pi = \pi \circ \delta_{BS}$, where g is a global transition function of the cellular automaton CA and δ_{BS} is that of the discrete billiard system BS.*

Proof. We assume that $\# \notin L$. Let Q be $Q = \{q_\#\} \cup \{q_{0a}, q_{1a} \mid a \in L\} \cup \{q_{0a+1b} \mid a, b \in L\}$. A local transition function $f : Q^3 \to Q$ is defined by Table 2, where $q_{right} \in Q$.

Table 2. The local function of the CA.

$f(q_{left}, q_{center}, q_{right})$		q_{center}			
		$q_\#$	q_{0a}	q_{1a}	q_{0a+1r}
q_{left}	$q_\#$	$q_\#$	q_{0a}	$q_\#$	q_{0a}
	q_{0l}	$q_\#$	q_{0a}	$q_\#$	q_{0a}
	q_{1b}	q_{1b}	$q_{0a'+1b'}$	q_{1b}	$q_{0a'+1b'}$
	q_{0l+1b}	q_{1b}	$q_{0a'+1b'}$	q_{1b}	$q_{0a'+1b'}$

We define $\pi : 2^B \to Conf(Q)$ by

$$\pi(A)(x) = \begin{cases} q_\# & \text{if } pt(A, x) = \phi, \\ q_{0a} & \text{if } pt(A, x) = \{(0, a)\}, \\ q_{1a} & \text{if } pt(A, x) = \{(1, a)\}, \\ q_{0a+1b} & \text{if } pt(A, x) = \{(0, a), (1, b)\}. \end{cases}$$

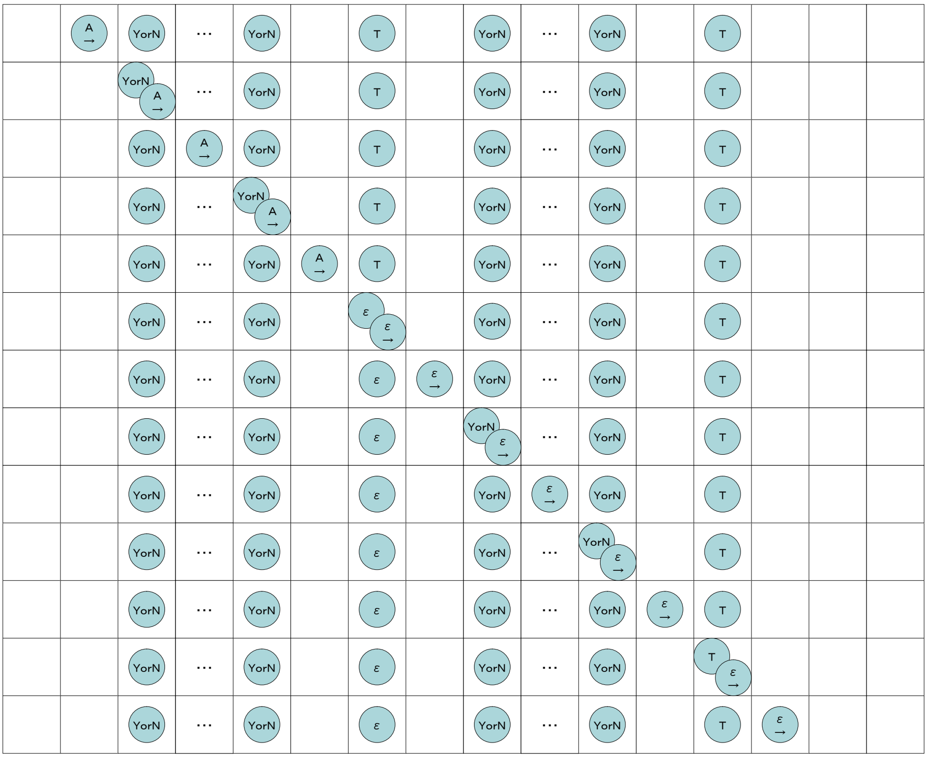


Fig. 6. A ball with label A passes through all balls with label Y and N, and delete a ball with label T, which is a terminal symbol of word.

It is easy to show that $g \circ \pi = \pi \circ \delta_{BS}$.

$$\begin{tikzcd}
2^B(BS) \arrow[r, "\pi"] \arrow[d, "\delta_{BS}"'] & Conf(Q)(CA) \arrow[d, "g"] \\
2^B \arrow[r, "\pi"'] & Conf(Q)
\end{tikzcd}$$

4 Cyclic tag systems and ACS

In this section, we study a discrete billiard system and a cyclic tag system. We show that a discrete billiard system can simulate any cyclic tag systems.

Definition 4.1. *Let Σ be a finite set. We call $w = w_1 w_2 \cdots w_n$ word on Σ. We denote n by $Len(w)$, which is called the length of w. Let λ be a word with $Len(\lambda) = 0$. We denote Σ^* by $\Sigma^* = \{w \mid w \text{ is a word on } \Sigma\}$.*

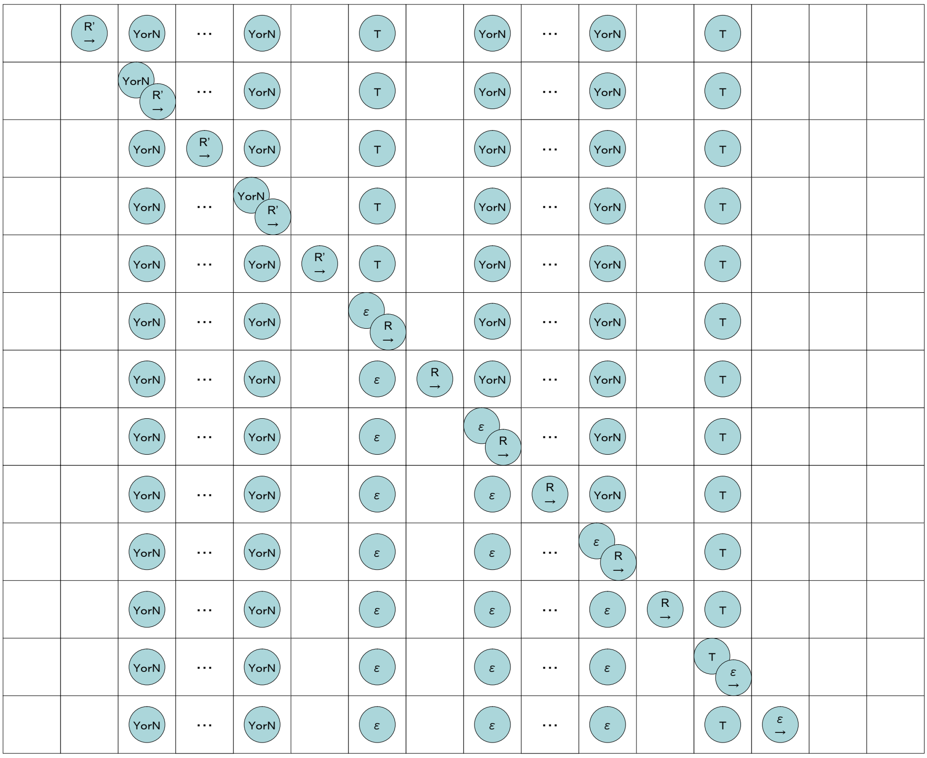

R' → YorN ··· YorN T YorN ··· YorN T
YorN R' → ··· YorN T YorN ··· YorN T
YorN R' → YorN T YorN ··· YorN T
YorN ··· YorN R' → T YorN ··· YorN T
YorN ··· YorN R' → T YorN ··· YorN T
YorN ··· YorN ε R → YorN ··· YorN T
YorN ··· YorN ε R → YorN ··· YorN T
YorN ··· YorN ε ε R → ··· YorN T
YorN ··· YorN ε ε R → YorN T
YorN ··· YorN ε ε ··· ε R → T
YorN ··· YorN ε ε ··· ε R → T
YorN ··· YorN ε ε ··· ε T ε →
YorN ··· YorN ε ε ··· ε T ε →

Fig. 7. A ball with label R' passes through all balls with label Y and N, and it produces the ball with label R when it collides a first ball with label T. The ball with label R sweeps out all balls with Y and N until next the ball with label T. So all balls between first and second balls with label T are deleted (changed into balls with label ε).

Definition 4.2. *We define a function obs* : $ACS \to L^*$. *It arranges labels of all balls sorted by their position. That is,* $X = \{(v_i, a_i, x_i) \mid a_i \in L, x_i \in \mathbb{Z}, i = 1, 2, \dots, n\}$, *and* $x_1 \leq x_2 \leq \cdots \leq x_n$, $x_i = x_{i+1} \Rightarrow v_i \leq v_{i+1}$. *then we define* $obs(S) = a_1 a_2 \cdots a_n$.

Definition 4.3 (A cyclic tag system). *A cyclic tag system is defined by* $C = (k, \{Y, N\}, (p_0, \dots, p_{k-1}))$, *where* $k \in \mathbb{N}$ *and* $(p_0, \dots, p_{k-1}) \in (\{Y, N\}^*)^k$. *A pair* (w, m) *is called a configuration of* C, *where* $w \in \{Y, N\}^*$ *and* $i \in \{0, 1, \dots, k-1\}$.
We denote the set of all configurations of C *by* $Conf_{CTS}(C) = \{Y, N\}^* \times \{0, 1, \dots, k-1\}$. *A transition function* $\delta_C : Conf_{CTS}(C) \to Conf_{CTS}(C)$ *is defined by*

$$\delta_C(Yw', m) = (w'p_m, m + 1 \mod k),$$
$$\delta_C(Nw', m) = (w', m + 1 \mod k).$$

Example 4.1. Consider a cyclic tag system $C = (2, \{Y, N\}, (YYY, N))$ and initial configuration $(Y, 0)$. Then we see that

$$\begin{aligned}
{\delta_C}^1(Y, 0) &= (YYY, 1),\\
{\delta_C}^2(Y, 0) &= (YYN, 0),\\
{\delta_C}^3(Y, 0) &= (YNYYY, 1).
\end{aligned}$$

We consider Example 2.3 again. By arranging balls to some special form, this BS can simulate a CTS. From now on, let B, $\mathcal{F}[B]$ and $f_{\mathcal{F}[B]} : \mathcal{F}[B] \to 2^B$ be those of Example 2.3. Let BS be a discrete billiard system $BS = (B, \mathcal{F}[B], f_{\mathcal{F}[B]})$. We define E_B by $E_B := \{(v, \varepsilon, x) \mid v \in V, x \in \mathbb{Z}\}$.

Lemma 4.1. *We have* $\delta_{BS}(X) \setminus E_B = \delta_{BS}(X \setminus E_B) \setminus E_B$.

Proof. Let $E_B^X := E_B \cap X$ and $X' = X \setminus E_B^X$, then we have $\delta_{BS}(X) = \delta_{BS}(X' \cup E_B^X) = \delta_{BS}(X') \cup \delta_{BS}(E_B^X)$. Since $E_B^X \subseteq E_B$ and $\delta_{BS}(E_B^X) \subseteq E_B$, then we have $\delta_{BS}(X') \cup \delta_{BS}(E_B^X) \setminus E_B = \delta_{BS}(X') \setminus E_B$. Hence $\delta_{BS}(X) \setminus E_B = \delta_{BS}(X') \setminus E_B = \delta_{BS}(X \setminus E_B) \setminus E_B$.

Definition 4.4. *Let C be a cyclic tag system $C = (k, \{Y, N\}, (p_0, \ldots, p_{k-1}))$ and $(w, m) \in \{Y, N\}^* \times \{0, 1, \ldots, k-1\}$, We denote $P_i \in \{Y, N\}^*$ by $P_i = p_{(m+i \mod k)}$. For $X \in 2^B$, we define $X \in ARR(C, w, m)$ by*

$$\begin{aligned}
&X \setminus E_B\\
&= \left\{(1, H, x_H^{(i)}) \mid i = 1, 2, \ldots\right\}\\
&\bigcup \left\{(0, w_j, x_j^{(0)}) \mid w = w_1 \cdots w_n, j = 1, 2, \ldots, n\right\}\\
&\bigcup \left\{(0, T, x_T^{(0)})\right\}\\
&\bigcup \left\{ \begin{matrix} (0, (P_i)_j, x_j^{(i)}), \\ (0, T, x_T^{(i)}) \end{matrix} \,\middle|\, \begin{matrix} i = 1, 2, \ldots, \\ P_i = (P_i)_1 (P_i)_2 \cdots (P_i)_{n_i}, j = 1, 2, \ldots, n_i \end{matrix} \right\},
\end{aligned} \tag{7}$$

where $x_j^{(i)}$ satisfies

$$\begin{gathered}
x_j^{(i)} > 0 \text{ for any } i, j,\\
x_j^{(i)} < x_{j+1}^{(i)} < x_T^{(i)},\\
x_T^{(i)} < x_0^{(i+1)}.
\end{gathered} \tag{8}$$

Moreover, $x_H^{(i)} (i = 1, 2, \ldots)$ are determined by the following recursive formula:

$$x_H^{(1)} = 0, x_H^{(i+1)} = x_H^{(i)} - x_T^{(i)}. \tag{9}$$

We put

$$\begin{aligned}
\mathit{head}(X, i) &:= x_H^{(i)} \text{ for } i = 1, 2, \ldots,\\
\mathit{term}(X, i) &:= x_T^{(i)} \text{ for } i = 0, 1, 2, \ldots.
\end{aligned} \tag{10}$$

Remark 4.1. By using Eq. (10), we can rewrite Eq. (9) by

$$\begin{aligned} head(X,1) &= 0, \\ head(X,i+1) &= head(X,i) - term(X,i). \end{aligned} \tag{11}$$

Lemma 4.2. *Let C be a cyclic tag system, (w,m) be a configuration of C. Then it follows that $X,Y \in ARR(C,w,m) \Rightarrow obs(X) = obs(Y)$.*

Proof. Since $X,Y \in ARR(C,w,m)$, both X and Y satisfy Eq. (7) and Eq. (8). Therefore Definition 4.2 shows this lemma.

Lemma 4.3. *Let C be a cyclic tag system, (w_0,m_0) be a configuration of C. Assume that $X \in ARR(C,w_0,m_0)$. Put $T = term(X,1) - head(X,1)$. Then we have*

(1) $X' := {\delta_{BS}}^T(X) \in ARR(C,w_1,m_1)$,
(2) $head(X',i) = head(X,i+1) + T$ *for* $i = 1,2,\ldots$,
(3) $term(X',i) = term(X,i+1)$ *for* $i = 0,1,2,\ldots$,

where $(w_1,m_1) = \delta_C((w_0,m_0))$.

Proof. It is sufficient to consider the two cases of $w_0 = Yw'$ and $w_0 = Nw'$. At first, we show the former. Since $X \in ARR(C,Yw',m_0)$, we can write

$$\begin{aligned} & X \setminus E_B \\ = & \left\{(1,H,x_H^{(i)}) \mid i = 1,2,\ldots\right\} \\ \bigcup & \left\{(0,Y,x_1^{(0)})\right\} \\ \bigcup & \left\{(0,w'_j,x_j^{(0)}) \,\middle|\, \begin{array}{c} w' = w'_1 w'_2 \cdots w'_{n'}, \\ j = 1,2,\ldots,n' \end{array}\right\} \\ \bigcup & \left\{(0,T,x_T^{(0)})\right\} \\ \bigcup & \left\{(0,(P_i)_j,x_j^{(i)}),(0,T,x_T^{(i)}) \,\middle|\, \begin{array}{c} i = 1,2,\ldots, \\ P_i = (P_i)_1 \cdots (P_i)_{n_i}, j = 1,2,\ldots,n_i \end{array}\right\}. \end{aligned}$$

We have $T = term(X,1) - head(X,1) = x_T^{(1)} - x_H^{(1)}$. Note that Remark 2.2 provides us with

$$\begin{aligned} & {\delta_{BS}}^T(X) \setminus E_B \\ = & \left\{(1,H,x_H^{(i+1)} + T) \mid i = 1,2,\ldots\right\} \\ \bigcup & \left\{(0,w'_j,x_j^{(0)}) \mid w' = w'_1 \cdots w'_{n'} j = 1,2,\ldots,n'\right\} \\ \bigcup & \left\{(0,(P_1)_j,x_j^{(1)}),(0,T,x_T^{(1)}) \,\middle|\, \begin{array}{c} P_1 = (P_1)_1 \cdots (P_1)_{n_1}, \\ j = 1,2,\ldots,n_1 \end{array}\right\} \\ \bigcup & \left\{\begin{array}{c} (0,(P_{i+1})_j,x_j^{(i)}), \\ (0,T,x_T^{(i+1)}) \end{array} \,\middle|\, \begin{array}{c} i = 1,2,\ldots, \\ P_{i+1} = (P_{i+1})_1 \cdots (P_{i+1})_{n_{i+1}}, \\ j = 1,2,\ldots,n_{i+1} \end{array}\right\}. \end{aligned}$$

Since $(w_1, m_1) = (w'P_1, m_0 + 1 \mod k)$, we get

$$\begin{aligned}
&{\delta_{BS}}^{T}(X) \setminus E_B \\
&= \left\{(1, H, x_H^{(i+1)} + T) \mid i = 1, 2, \ldots\right\} \\
&\bigcup \left\{(0, (w_1)_j, x_j^{(0)}) \mid w_1 = (w_1)_1 \cdots (w_1)_{n_1}, j = 1, 2, \ldots, n_1\right\} \\
&\bigcup \left\{(0, T, x_T^{(1)})\right\} \\
&\bigcup \left\{ \begin{array}{c} (0, (P_{i+1})_j, x_j^{(i)}), \\ (0, T, x_T^{(i+1)}) \end{array} \middle| \begin{array}{c} i = 1, 2, \ldots, \\ P_{i+1} = (P_{i+1})_1 \cdots (P_{i+1})_{n_{i+1}}, \\ j = 1, 2, \ldots, n_{i+1} \end{array} \right\}.
\end{aligned}$$

Hence

$$\begin{aligned} head(X', i) &= x_H^{(i+1)} + T = head(X, i+1) + T, \\ term(X', i) &= x_T^{(i+1)} = term(X, i+1) \end{aligned} \tag{12}$$

(the second and third statements of lemma). Moreover, it is easy to show that $head(X', 1) = 0$ and $head(X', i+1) = head(X', i) - term(X', i)$. Hence we can conclude that $X' = {\delta_{BS}}^{T} \in ARR(C, w_1, m_1)$. By a similar way, we can show the latter.

Theorem 4.1. *Let C be a cyclic tag system and $X_0 \in ARR(C, w_0, m_0)$. Then, we have $\delta_{BS}^{T(t)}(X_0) \in ARR(C, w_t, m_t)$,*
where

$$T(0) = 0, \tag{13}$$

$$T(t) = term(X_0, t) - head(X_0, t), \tag{14}$$

$$(w_t, m_t) = {\delta_C}^{t}((w_0, m_0)).$$

Proof. We show this by using mathematical induction. In the case of $t = 0$, it is clear. Let $X_t := {\delta_{BS}}^{T(t)}(X_0)$. We assume that

(1) $X_t \in ARR(C, v_t, m_t)$,
(2) $head(X_t, i) = head(X_0, t+i) + T(t)$ and
(3) $term(X_t, i) = term(X_0, t+i)$.

It is sufficient to show that

(1) $X_{t+1} \in ARR(C, w_{t+1}, m_{t+1})$,
(2) $head(X_{t+1}, i) = head(X_0, t+1+i) + T(t+1)$ and
(3) $term(X_{t+1}, i) = term(X_0, t+1+i)$.

Since

$$\begin{aligned} T &= term(X_t, 1) - head(X_t, 1) \\ &= term(X_0, t+1) - (head(X_0, t+1) + T(t)) \\ &= (term(X_0, t+1) - (head(X_0, t+1)) - T(t) \\ &= T(t+1) - T(t), \end{aligned}$$

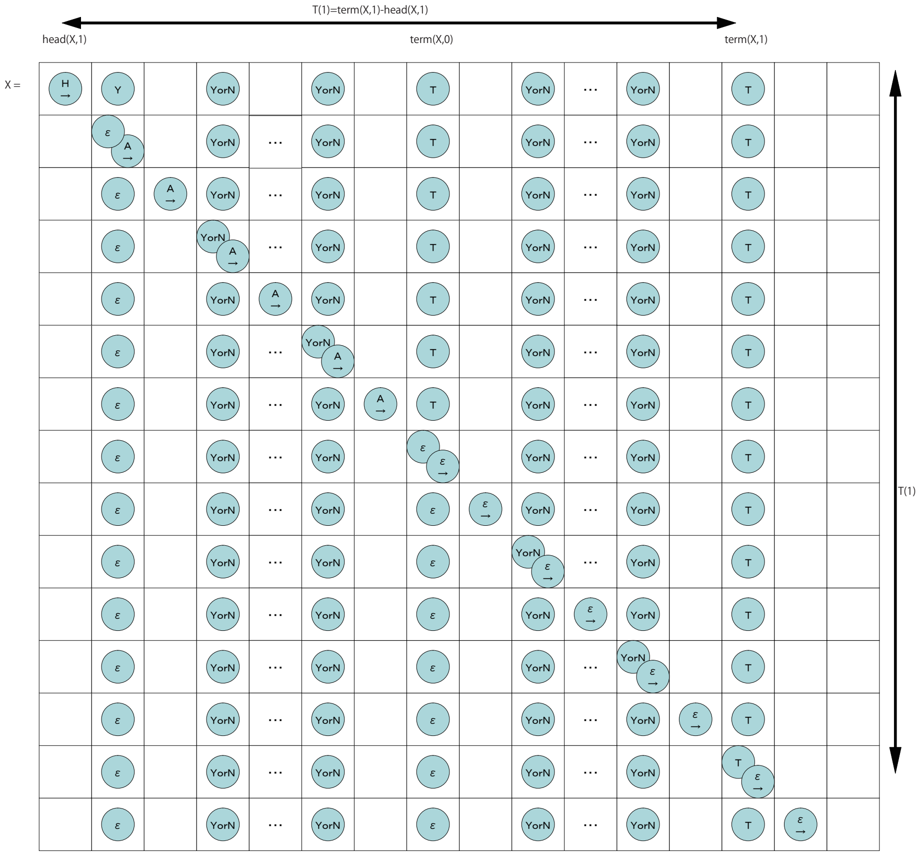


Fig. 8. A discrete billiard system simulates 1 step of a cyclic tag system. This figure shows the proof of the former case of Lemma 4.3.

we have

$$\begin{aligned} X_{t+1} &= \delta_{BS}^{T(t+1)}(X_0) \\ &= \delta_{BS}^{T(t+1)-T(t)+T(t)}(X_0) \\ &= \delta_{BS}^{T(t+1)-T(t)} \circ \delta_{BS}^{T(t)}(X_0) \\ &= \delta_{BS}^{T}(X_t) \in ARR(C, w_{t+1}, m_{t+1}) \text{ (by Lemma 4.3) .} \end{aligned}$$

Furthermore, it follows that $head(X_{t+1}, i) = head(X_t, i+1)+T = (head(X_0, (t+i)+1)+T(t))+T = head(X_0, t+1+i)+T(t+1)$ and $term(X_{t+1}, i) = term(X_t, i+1) = term(X_0, t+i+1)$.

Corollary 4.1. *For each cyclic tag system C, we consider a function $Conf_{CTS}(C) \to 2^B$ which satisfies $in_C((w, m)) \in ARR(C, w, m)$. It is not determined uniquely. However, we choose one of such functions and fix it. Let*

(w, m) be a configuration of C. Then we have

$$obs \circ {\delta_{BS}}^{T(t)} \circ in_C((w,m)) = obs \circ in_C \circ {\delta_C}^t((w,m)), \tag{15}$$

where δ_{BS} is the global transition function of BS, δ_C is the transition function of C and $T(t)$ is formulated by

$$\begin{aligned} X_0 &= in_C(w,m), \\ T(0) &= 0, \\ T(t) &= term(X_0,t) - head(X_0,t). \end{aligned} \tag{16}$$

Proof. Theorem 4.1 shows that ${\delta_{BS}}^{T(t)} \circ in_C((w,m)) \in ARR(C, w', m')$ and $in_C \circ {\delta_C}^t((w,m)) \in ARR(C, w', m')$, where $(w', m') = {\delta_C}^t((w,m))$. By Lemma 4.2,it follows Eq. (15).

$$\begin{tikzcd}
Conf_{CTS}(C) \arrow[r, "in"] \arrow[d, "{\delta_C}^t"'] & 2^B \arrow[d, "{\delta_{BS}}^{T(t)}"] & \\
Conf_{CTS}(C) \arrow[r, "in"'] & 2^B \arrow[r, "obs"] & L^*
\end{tikzcd}$$

We give a example of the simulation.

Example 4.2. For a given cyclic tag system $C = (2, \{Y, N\}, (YYY, N))$ and initial configuration $(Y, 0)$, we have

$$in_C(Y,0) = \left\{ \begin{array}{c} \ldots, (1, H, -x_6 - x_8), (1, H, -x_6), (1, H, 0), \\ (0, Y, x_1), (0, T, x_2), \\ (0, Y, x_3), (0, Y, x_4), (0, Y, x_5), (0, T, x_6), \\ (0, N, x_7), (0, T, x_8), \\ (0, Y, x_9), (0, Y, x_{10}), (0, Y, x_{11}), (0, Y, x_{12}), \ldots \end{array} \right\}.$$

By the expression of $T(t)$, we have $T = x_6 - 0 = x_6$. Hence we can compute

$$\begin{aligned} &{\delta_{BS}}^T(in_C(Y,0)) \\ &= \left\{ \begin{array}{c} \ldots, (1, H, -x_8), (1, H, 0), (0, \varepsilon, x_1), (0, \varepsilon, x_2), \\ (0, Y, x_3), (0, Y, x_4), (0, Y, x_5), (0, T, x_6), (0, \varepsilon, x_6), \\ (0, N, x_7), (0, T, x_8), \\ (0, Y, x_9), (0, Y, x_{10}), (0, Y, x_{11}), (0, T, x_{12}), \ldots \end{array} \right\}. \end{aligned}$$

Therefore, the left hand side is $\cdots HHYYYTNTYYYT \cdots$.
On the other hand, we can compute

$$\begin{aligned} in_C \circ \delta_C((Y,0)) &= in_C((YYY,1)) \\ &= \left\{ \begin{array}{c} \ldots, (1, H, -y_4), (1, H, 0), \\ (0, Y, y_1), (0, Y, y_2), (0, Y, y_3), (0, T, y_4), \\ (0, N, y_5), (0, T, y_6), \\ (0, Y, y_7), (0, Y, y_8), (0, Y, y_9), (0, T, y_{10}), \ldots \end{array} \right\}. \end{aligned}$$

Hence we can see that the right hand side is
$\cdots HHYYYTNTYYYT \cdots$. Therefore, we get Eq. (15).

Corollary 4.2. *Let C be a cyclic tag system. There exists a cellular automaton $CA = (Q, f)$ and $\bar{\pi} : \{Y, N\}^* \times \{0, 1, \dots, k-1\} \rightarrow Conf(Q)$ such that $g \circ \bar{\pi} = \bar{\pi} \circ \delta_C$, where g is a global transition function of the cellular automaton CA and δ_C is that of the cyclic tag system C.*

5 Concluding remarks

In this paper, we defined a new computation model named an abstract collision system, and defined a discrete billiard system as a special case of an abstract collision system. We studied the relation between the model and another computation models, such as a cellular automaton and a cyclic tag system. We proved that a discrete billiard system could simulate any cyclic tag system. Since a cyclic tag system is universal for computation, so does a discrete billiard system. Moreover, we proved that there was a natural correspondence between a discrete billiard system (with some conditions) and a cellular automaton. More precisely, a cellular automaton can simulate a discrete billiard system which simulates a cyclic tag system. This means that a cellular automaton can simulate a cyclic tag system. Hence we can conclude that a cellular automaton is universal for computation.

References

[1] A. Adamatzky (Ed)., *Collision-Based Computing*, Springer, 2002.

[2] E.R. Berlekamp, J.H.Conway and R.Guy, *Winning Ways for Your Mathematical Plays*, vol 2, Academic Press, 1982.

[3] M. Cook. *Universality in elementary cellular automata*, Complex Systems, 15, p1-40 (2004).

[4] K. Morita. *Simple universal one-dimensional reversible cellular automata*, Journal of Cellular Automata, Vol 2, pp 159-166 (2007).

[5] S. Wolfram. *A New Kind of Science.* (Wolfram Media, 2002).

Computation by competing patterns: Life rule $B2/S2345678$

Genaro J. Martínez[1], Andrew Adamatzky[1], Harold V. McIntosh[2], and
Benjamin De Lacy Costello[3]

[1] Faculty of Computing, Engineering and Mathematical Sciences, University of the West of England, Bristol, United Kingdom
{genaro.martinez,andrew.adamatzky}@uwe.ac.uk
http://uncomp.uwe.ac.uk/genaro/

[2] Departamento de Aplicación de Microcomputadoras, Instituto de Ciencias, Universidad Autónoma de Puebla, Puebla, México.
mcintosh@servidor.unam.mx
http://delta.cs.cinvestav.mx/~mcintosh/

[3] Faculty of Applied Sciences, University of the West of England, Bristol, United Kingdom

Abstract. Patterns, originating from different sources of perturbations, propagating in a precipitating chemical medium do usually compete for the space. They sub-divide the medium onto the regions unique for an initial configuration of disturbances. This sub-division can be expressed in terms of computation. We adopt an analogy between precipitating chemical media and semi-totalistic binary two-dimensional cellular automata, with cell-state transition rule $B2/S2\dots8$. We demonstrate how to implement basic logic and arithmetical operations, i.e. computability, by patterns propagating in geometrically constrained Life rule $B2/S2\dots8$ medium.

1 Introduction

Non-standard computation deals with implementation of programmable processing of information by unorthodox ways (e.g. computing with traveling localizations [2]) and in novel, or unusual, materials, (e.g. chemical reaction-diffusion media [4]). All non-standard computational systems can be classified as geometrically-constrained (fixed, stationary, architecture, e.g. wires, gates) and architectureless (collision-based, or 'free space'[4] [1]) computers.

Conway's Game of Life [11] is the best-known example of a universal collision-based computer [8, 2]. Its universality [18] can be proved and demonstrated by many ways, either simple functionally complete set of logical functions, as in [8], or via construction of large and complicated simulators of Turing machine [9, 25] and register machine [8].

[4] 'Free space computing' is a term coined by Jonathan Mills.

The Life, and Life-like rules, are know to support myriad of traveling localizations, or gliders; stationary localizations, or still lives; breathing stationary localizations, or oscillators [11, 29, 12, 20, 24, 5, 30]. In its original form, where transition from living state, '1', to 'death' state, '2', does exist, the Life automata resemble excitable media, including excitable reaction-diffusion systems. There is also a family of Life-life rules, where 'cells never die', or the state '1' is an absorbing state. This is the family of *Life without Death*, invented by Griffeath and Moore in [12]. In the Life without Death automata we can still observe propagating localizations, formed due to rule-based restrictions on propagation similar to that in sub-excitable chemical media and plasmodium of *Physarum polycephalum* [7], but no complicated periodic structures or global chaotic behavior occurs.

The Life without Death family of cell-state transition rules is the Game of Life equivalent of the precipitating chemical systems. This is demonstrated in our computational-phenomenological studies of semi-totalistic and precipitating CA [5], where we selected a set of rules, identified by periodic structures, which is named as Life $2c22$ [21].[5] The clans closest to the family $2c22$ are *Diffusion Rule* (Life rule $B2/S7$) [17], these clans also belong to the big cluster known as Life $dc22$.

The Life families with indestructible patterns allow us to study a computational potential of the propagating precipitating systems. We employ our previous results on chemical laboratory prototypes of XOR gates in reaction-diffusion chemical media [3], and design a binary adder in the CA equivalent of the precipitating medium. In Sect. 2 we overview basic patterns emerging in rule $B2/S2\dots8$. Logical gates and a binary full adder are constructed in Sect. 3.

2 Life rule $B2/S2\dots8$

The Life rule $B2/S2\dots8$ is described as follows. Each cell takes two states '0' ('dead') and '1' ('alive'), and updates its state depending on its eight closest neighbors as follows:

1. Birth: a central cell in state 0 at time step t takes state 1 at time step $t+1$ if it has exactly two neighbors in state.
2. Survival: a central cell in state 1 at time t remains in the state 1 at time $t+1$ if it has more then one live neighbor.
3. Death: all other local situations.

Once a resting lattice is perturbed, few cells assigned live states, patterns formed and grow quickly. Most interesting behavior occurs when at least 20% of cells are initially alive. A general behaviour of rule $B2/S2\dots8$ can be well described by a mean field polynomial and its fixed points (see Fig. 1) [23, 14], as follow:

[5] http://uncomp.uwe.ac.uk/genaro/diffusionLife/life_2c22.html

$$p_{t+1} = 28p_t^2 q_t^7 + 28p_t^3 q_t^6 + 56p_t^4 q_t^5 + 70p_t^5 q_t^4 + 56p_t^6 q_t^3 + 28p_t^7 q_t^2 + 8p_t^8 q_t + p_t^9.$$

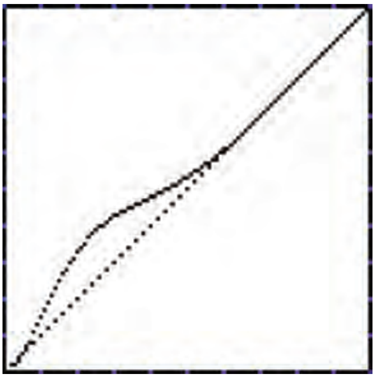


Fig. 1. Mean field curve for $B2/S2\ldots8$.

High densities of domains dominated by state 1 correspond to $p = 0.9196$ to $p = 1$ (also we can consider $p = 0.6598$ to $p = 0.7252$, all they are stable fixed points). Interesting behavior can be found in extreme unstable fixed points when $p = 0.00036$ to $p = 0.001$. The unstable fixed points may represent gliders and small oscillators.

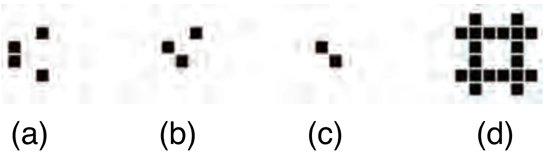


Fig. 2. Basic periodic structures in $B2/S2\ldots8$: (a) glider period one, (b) oscillator period one, (c) flip-flop, and (d) still life configuration.

Minimal localizations, or basic periodic structures, in rule $B2/S2\ldots8$ include gliders, oscillators, flip-flops, and still life (stationary localization) configurations (Fig. 2).

A relevant characteristic was that the rule $B2/S2\ldots8$ supports *indestructible patterns*, which can not be destroyed from any perturbation, they belong to the class of stationary localizations, still lives [10, 22]. The minimal indestructible pattern is show in Fig. 2d. More heavier, in a number of live cells, patterns are provided in Fig. 3.

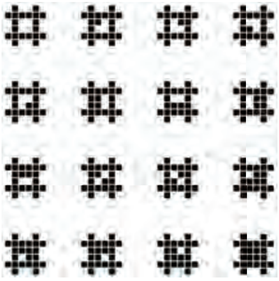


Fig. 3. Indestructible Still Life family patterns derived in rule $B2/S2 \ldots 8$.

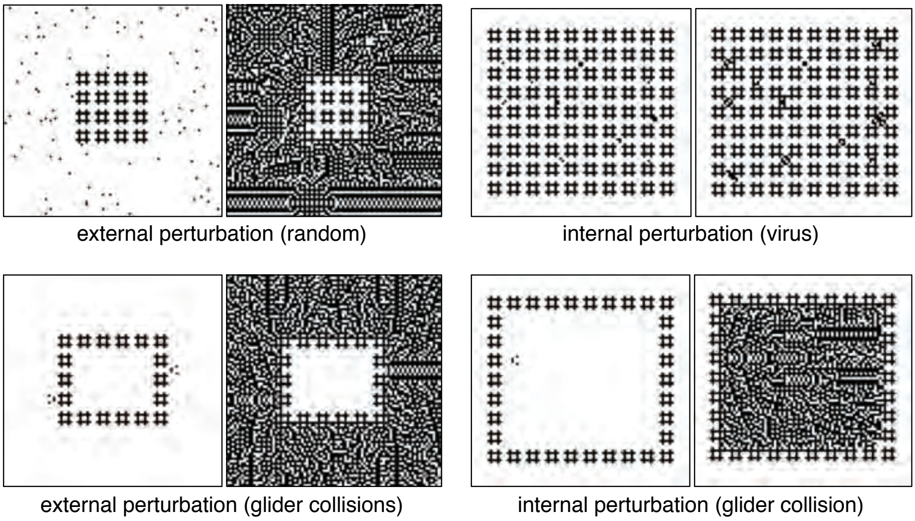

external perturbation (random)
internal perturbation (virus)
external perturbation (glider collisions)
internal perturbation (glider collision)

Fig. 4. Indestructible Still Life colonies 'tested' by internal and external perturbations. Each pair of snapshots represents an initial condition (on the left) and a final, i.e. stationary, configuration (on the right).

In CA rule $B2/S2 \ldots 8$ one can setup colonies of the indestructible structures as sets of block patterns, which are capable for resisting internal and external perturbations, see examples in Fig. 4. The indestructible patterns symbolize a precipitation in CA development. We use these patterns to architecture channels, or wires, for signal propagation.

The Still Life blocks are not affected by their environment however they do affect their environment. As demonstrated in Fig. 4, bottom scenarios, gliders colliding to the Still Life walls are transformed into propagating patterns, which fill the automaton lattice.

Localizations colliding to Still Life blocks become delocalised.

We use this feature of interactions between stationary and mobile localizations in designing logical circuits.

3 Computing with propagating patterns

We implement computation with patterns propagating in the Life rule $B2/S2\dots8$ is follows. A computing scheme is build as channels, geometrically constrained by Still Life indestructible blocks, and T-junctions[6] between the channels. Each T-junction consists of two horizontal channels A and B (shoulders), acting as inputs, and a vertical channel, C, assigned as an output. Such type of circuitry have been already used to implement XOR gate in chemical laboratory precipitating reaction-diffusion systems [3, 4], and precipitating logical gates in CA [16]. A minimal width of each channel is calculated as three widths of the Still Life block (Fig. 2d) and width of a glider (Fig. 2a).

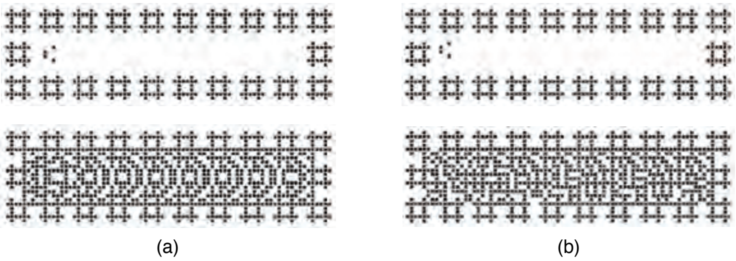


Fig. 5. Feedback channels constructed with still life patterns (a) show the initial state with the empty channel and a glider (top) and final state representing value 0 (low), and (b) show non-symmetric patterns representing value 1.

Boolean values are represented by gliders, positioned initially in the middle of channel, value 0 (Fig. 5a, top), or slightly offset, value 1 (Fig. 5b, top). The initial positions of the gliders determine outcomes of their delocalisation. Glider, corresponding to the value 0 delocalised into regular identified by a symmetric pattern, like frozen waves of excitation, patterns (Fig. 5a, bottom). Glider, representing the value signal value 1, delocalises into the less regular patterns (Fig. 5b, bottom) identified by a non-symmetric pattern although eventually it became periodic on a long channel but not symmetric.

The patterns, representing values 0 and 1, propagate along the channels and meet at the T-junctions. They compete for the output channel, and, depending on initial distance between gliders, one of the patterns wins and propagates along the output channel. Figure 6 shows final configurations of basic logical gates.

[6] T-junction based control signals were suggested also in von Neumann [28] works.

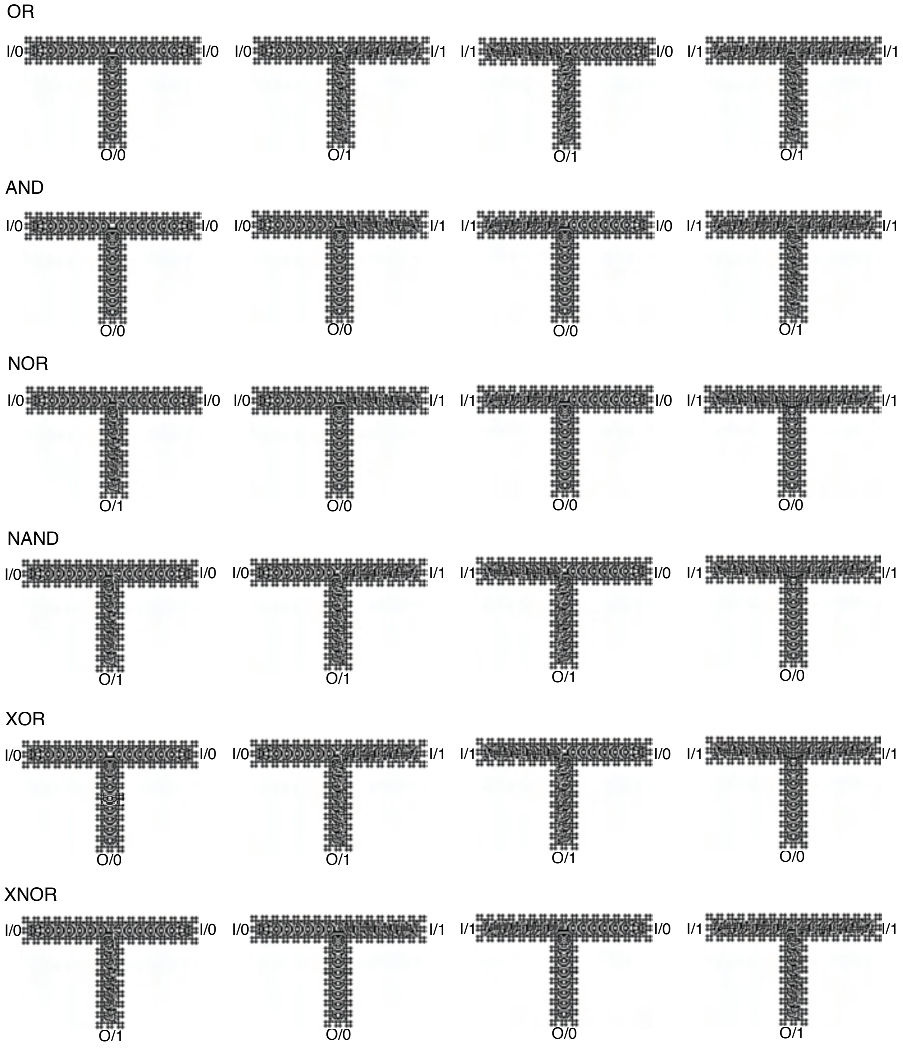


Fig. 6. Logic gates implemented at the Life rule $B2/S2\ldots8$. Input binary values A and B are represented for In/0 or In/1, output result C is represented by Out/0 or Out/1.

The gates can be cascaded into more 'useful' circuits, e.g. binary adders. See a scheme representation based in T-junctions from its traditional circuit of a binary half-adder in Fig. 7.

Final configurations of the one-bit half-adder are shown in Fig. 8. The circuity can be extended to a full adder (Fig. 9). Configurations of the adder, outlined with Still Life blocks, and description of computation stages, are shown in Fig. 10.

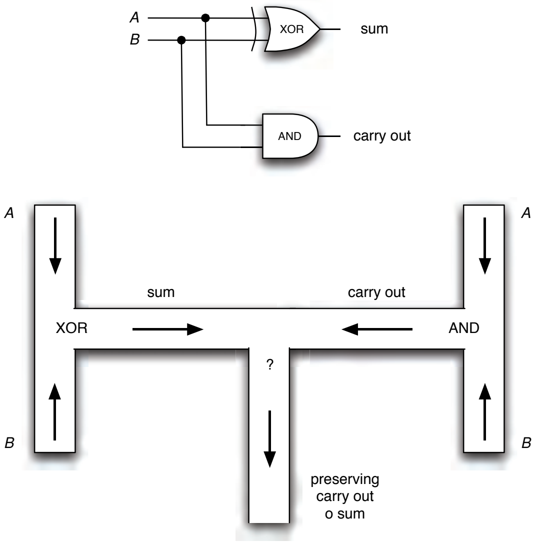


Fig. 7. Half adder circuit (top) and scheme of its implementation by propagating patterns geometrically constrained medium (bottom).

The full adder consists of 16 T-junctions, linked together by channels; signals are synchronized in the system.

The full adder occupies $1,118 \times 1,326$ cells lattice, and 66,630 cells are activated in 952 generations, during the computation. A data-area of the full adder is shown in Fig. 11.

4 Conclusions

We have demonstrated how to implement basic logical and arithmetical computation by propagating precipitating patterns[7] in geometrically constrained media. Also, we shown computation universality of Life rules, on example of the rule $B2/S2 \dots 8$, by implementing basic logical gates and binary adders. Source codes and specific initial condition (`.rle` files)[8] to reproduce and verify our results are available at `http://uncomp.uwe.ac.uk/genaro/DiffusionLife/B2-S2345678.html`

Future work will concern with explicit construction of a Turing machine, computer design, solitons [15], systems self-copying [19] and detailed study and classification of indestructible still life patterns in Life $dc22$. The implementations can also be analysed in a context of space complexity and different orders of

[7] The propagating patterns are generated by gliders; compare to the production of complex objects in Life during gliders' collisions `http://www.pentadecathlon.com/`

[8] Implementations and constructions were developed with Golly system available from `http://golly.sourceforge.net/`

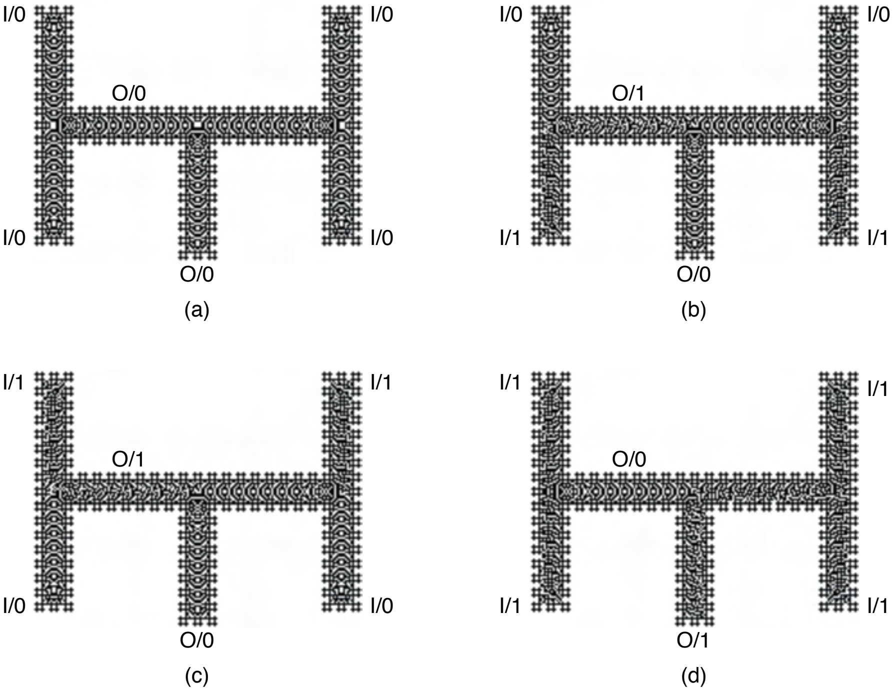


Fig. 8. Half adder implemented in *Life* rule $B2/S2\dots8$. Operations represent sums (a) $0+0$, (b) $0+1$, (c) $1+0$, and (d) $1+1$; CARRY OUT is preserved in this case.

CA [27], comparison to other complex consturctions in Life domain [9, 25], including the isotropic neighborhood [26].

Acknowledgement

This research was supported by Engineering and Physical Sciences Research Council (EPSRC), United Kingdom, grant EP/F054343.

References

[1] Adamatzky, A. (2001) *Computing in Nonlinear Media and Automata Collectives*, Institute of Physics Publishing, Bristol and Philadelphia.

[2] Adamatzky, A. (Ed.) (2003) *Collision-Based Computing*, Springer.

[3] Adamatzky A. & De Lacy Costello B. (2002) Experimental logical gates in a reaction-diffusion medium: The XOR gate and beyond, *Phys. Rev. E* **66**, 046112.

[4] Adamatzky, A., De Lacy Costello, B. & Asai, T. (2005) *Reaction-Diffusion Computers*, Elsevier.

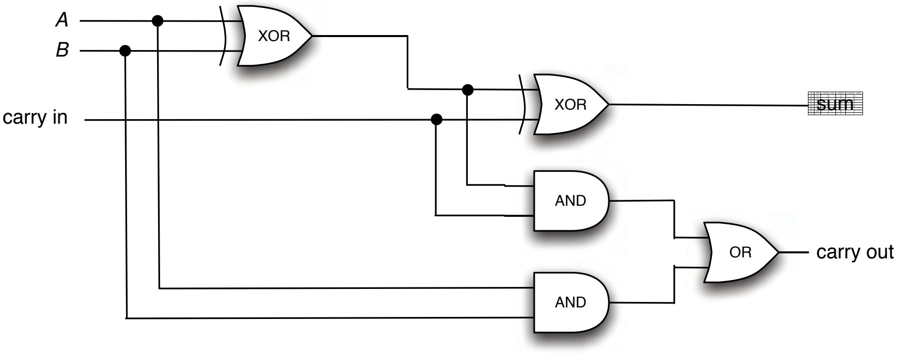


Fig. 9. Full binary adder circuit.

[5] Adamatzky, A., Martínez, G.J. & Seck Tuoh Mora, J.C. (2006) Phenomenology of reaction-diffusion binary-state cellular automata, *Int. J. Bifurcation and Chaos* **16 (10)**, 1–21.

[6] Adamatzky, A. & Wuensche, A. (2006) Computing in spiral rule reaction-diffusion cellular automaton, *Complex Systems* **16 (4)**.

[7] Adamatzky A., De Lacy Costello B., Shirakawa T. Universal computation with limited resources: Belousov-Zhabotinsky and Physarum computers. Int. J. Bifurcation and Chaos (2008), in press.

[8] Berlekamp, E.R., Conway, J.H. & Guy, R.K. (1982) *Winning Ways for your Mathematical Plays*, Academic Press, (vol. 2, chapter 25).

[9] Chapman, P. (2002) Life Universal Computer, `http://www.igblan.free-online.co.uk/igblan/ca/`

[10] Cook, M. (2003) Still Life Theory, (in ref. [13]).

[11] Gardner, M. (1970) Mathematical Games — The fantastic combinations of John H. Conway's new solitaire game Life, *Scientific American* **223**, 120–123.

[12] Griffeath, D. & Moore, C. (1996) Life Without Death is P-complete, *Complex Systems* **10**, 437–447.

[13] Griffeath, D. & Moore, C. (Eds.) (2003) *New constructions in cellular automata*, Oxford University Press.

[14] Gutowitz, H.A. & Victor, J.D. (1987) Local structure theory in more that one dimension, *Complex Systems* **1**, 57–68.

[15] Jakubowski, M.H., Steiglitz, K. & Squier, R. (2001) Computing with Solitons: A Review and Prospectus, *Multiple-Valued Logic*, Special Issue on Collision-Based Computing, vol. 6, num. 5–6.

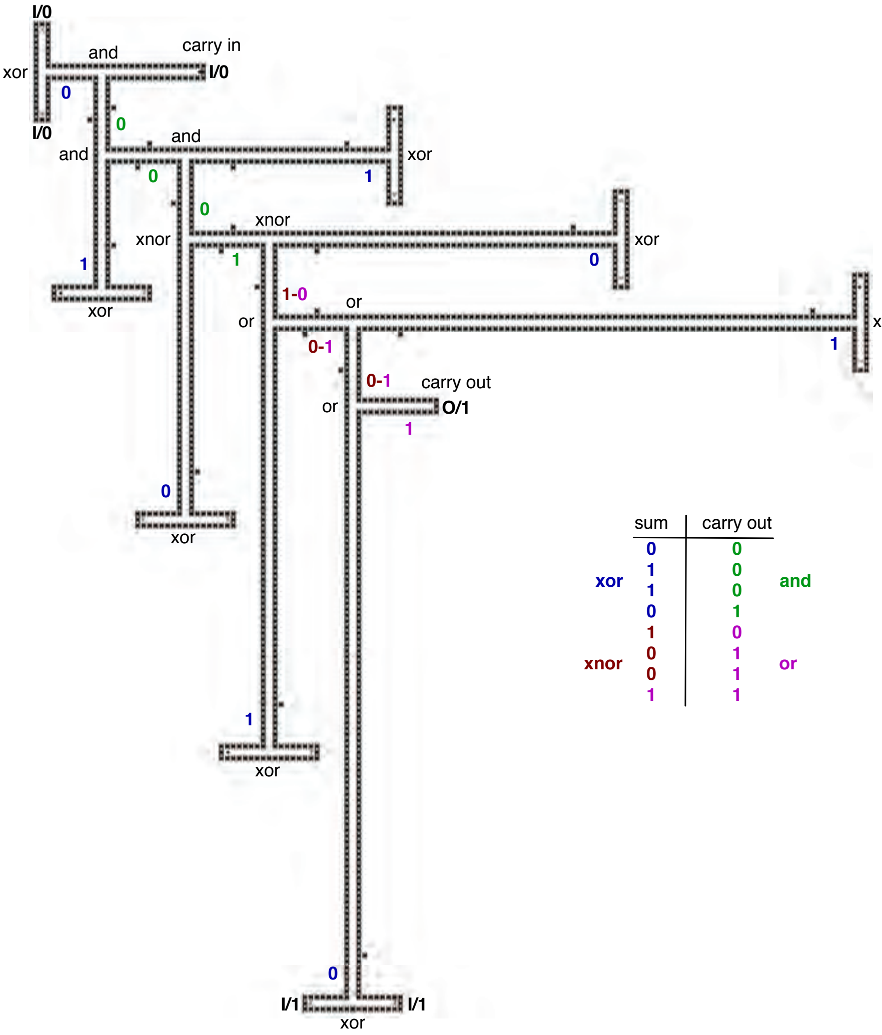
[16] Martínez, G.J., Adamatzky, A. & De Lacy Costello, B., On logical gates in precipitating medium: cellular automaton model, *Physics Letters A*, accepted.

[17] Martínez, G.J., Adamatzky, A. & McIntosh, H.V., Localization dynamic in a binary two-dimensional cellular automaton: the Diffusion Rule, *J. Cellular Automata*, in press.

[18] Minsky, M. (1967) *Computation: Finite and Infinite Machines*, Prentice Hall.

[19] Mitchell, M. (2001) Life and evolution in computers, *History and Philosophy of the Life Sciences* **23**, pages 361–383.

[20] Magnier, M., Lattaud, C. & Heudin, J.-K. (1997) Complexity Classes in the Two-dimensional Life Cellular Automata Subspace, *Complex Systems* **11 (6)**, 419–436.



	sum	carry out	
	0	0	
xor	1	0	and
	1	0	
	0	1	
	1	0	
xnor	0	1	or
	0	1	
	1	1	

Fig. 10. Full binary adder description of stages in Life rule $B2/S2\ldots8$.

[21] Martínez, G.J., Méndez, A.M. & Zambrano, M.M. (2005) Un subconjunto de autómata celular con comportamiento complejo en dos dimensiones, `http://uncomp.uwe.ac.uk/genaro/papers.html`

[22] McIntosh, H.V. (1988) Life's Still Lifes, `http://delta.cs.cinvestav.mx/~mcintosh`

[23] McIntosh, H.V. (1990) Wolfram's Class IV and a Good Life, *Physica D* **45**, 105–121.

[24] Packard, N. & Wolfram, S. (1985) Two-dimensional cellular automata, *J. Statistical Physics*, **38**, 901–946.

[25] Rendell, P. (2002) Turing universality of the game of life, (in ref. [2]).

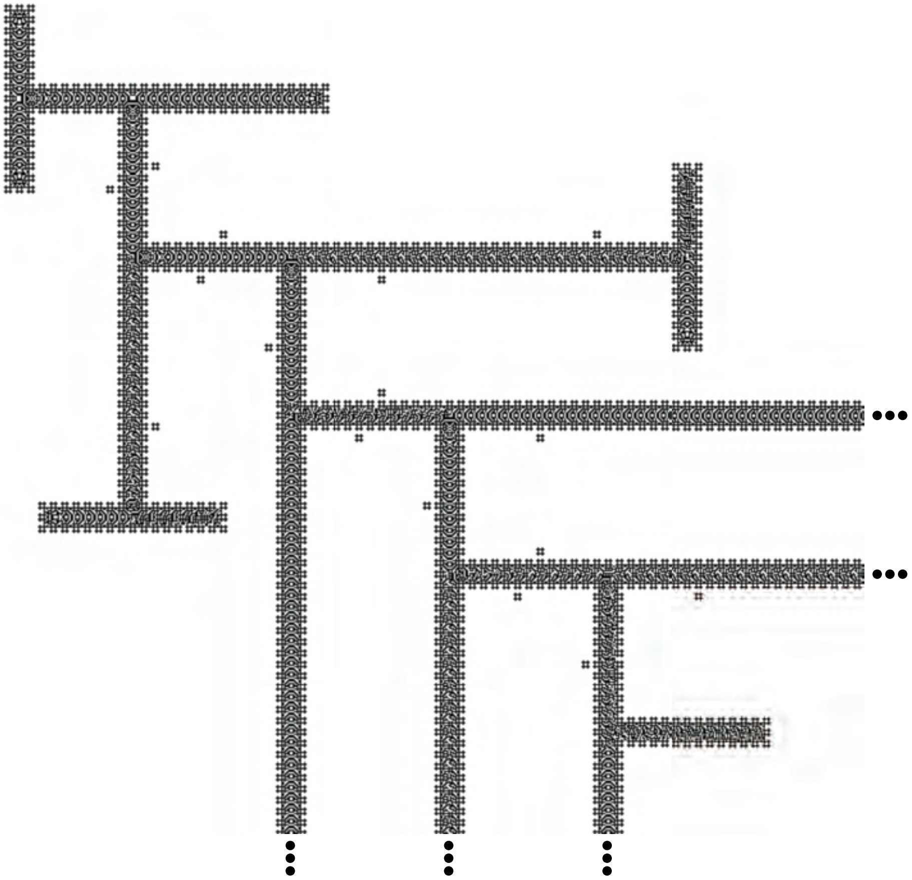


Fig. 11. Serial implementaton of the full binary adder in Life rule $B2/S2\dots8$, data area is enlarged.

[26] Sapin, E., Bailleux, O., Chabrier, J. & Collet, P. (2007) Demonstration of the Universality of a New Cellular Automaton, *Int. J. Unconventional Computing* **3**, 79–103.

[27] Tommaso, T. (2008) Background for the Quad Prize, EPSRC Automata 2008 Workshop, `http://uncomp.uwe.ac.uk/automata2008/prize.html`

[28] von Neumann, J. (1966) *Theory of Self-reproducing Automata* (edited and completed by A. W. Burks), University of Illinois Press, Urbana and London.

[29] Wainwright, R. (Ed.) Lifeline — A Quaterly Newsletter for Enthusiasts of John Conway's Game of Life, Issues 1 to 11, March 1971 to September 1973.

[30] Wuensche, A. (2004) Self-reproduction by glider collisions: the beehive rule, *Alife9 Proceedings*, 286–291, MIT Press.

Cellular automata sound synthesis: from histograms to spectrograms

Jaime Serquera and Eduardo R. Miranda

Interdisciplinary Centre for Computer Music Research (ICCMR),
University of Plymouth, Drake Circus, Plymouth, Devon PL4 8AA, UK
{jaime.serquera,eduardo.miranda}@plymouth.ac.uk

Abstract. We are investigating ways in which advanced unconventional computation methods may provide novel approaches to music technology. In this paper we report on a new technique to synthesise sounds from Cellular Automata (CA) models of reaction-diffusion chemical computers. We render the behaviour of CA into sounds using statistical analysis of CA cells as their values change in time. We estimate the probability distributions of the cells values for each cycle of the automaton by histogram measurements of the images produced by plotting the values of the CA cells as a matrix of coloured pixels. We have studied the behaviour of a CA model proposed by Gerhard and Schuster under a variety of different settings in order to gain a better understanding of the histograms, with a view on predicting the types of sounds that different CA behaviours would produce.

1 Introduction

The field of Computer Music is as old as Computer Science. Computers have been programmed to play music as early as the early 1950's when Geoff Hill programmed the CSIR Mk1 computer in Australia to play popular musical melodies [1]. Nowadays, the computer is becoming increasingly ubiquitous in all aspects of music. Uses of computer technology in music range from systems for musical composition to systems for distribution of music on the Internet. The implementation of such applications often demands the skillful combination of software engineering and artistic creativity. Whereas most current research into computers and music focuses on the development of media technology for delivering music to consumers (e.g., MP3 format, Internet search engines, and so on) our research focuses on the development of technology for musical creativity; that is, technology to aid musicians to create content for the media. We are particularly interested in investigating ways in which unconventional computation methods may provide novel approaches to music technology, particularly for the design of new musical instruments. In this paper we report on a new technique to synthesise sounds from Cellular Automata (CA) models of reaction-diffusion chemical computers [2].

In this paper, the CA model used to illustrate our new synthesis technique is based on an automaton proposed by Gerhard and Schuster [3], where states of a cell are interpreted metaphorically as follows: the state characterized by a minimum value 0 is called *healthy.* The state given by a maximum value $V-1$ is called *ill.* All other states in between are called *infected.* The transition rules are expressed as follows:

- RULE 1: IF $m_{x,y}[t] = 0$ THEN $m_{x,y}[t+1] = int(\frac{A}{r1}) + int(\frac{B}{r2})$
- RULE 2: IF $0 < m_{x,y}[t] < V-1$ THEN $m_{x,y}[t+1] = int(\frac{S}{A}) + K$
- RULE 3: IF $m_{x,y}[t] = V-1$ THEN $m_{x,y}[t+1] = 0$

where the value of a cell at a time step t is denoted by $m_{x,y}[t]$; x and y are the horizontal and vertical coordinates of the location of the cell in the CA grid. A and B represent the number of infected and ill cells in the neighbourhood, respectively; r_1 and r_2 are constants (which can be set to different values); S stands for the sum of the values of all cells in the neighbourhood; and V is the number of possible values that a cell can adopt. A desirable property of this CA model is its cyclic nature, which allows us to work with different ranges of cell values (also referred to in this paper as *colours*).

2 Rendering spectrograms from cellular automata histograms

In a nutshell, our method for rendering sounds from CA involves a mapping from CA histograms onto sound spectrograms. We devised a method to render the behaviour of CA into sounds using statistical analysis of the CA cells as their values change in time. We estimate the probability distributions of the cell values for each cycle of the automaton by histogram measurements [4] of the images generated by plotting the values of the CA cells as a matrix of coloured pixels. The histogram of a digital image with levels of gray colour in the range $[0, L-1]$ is a discrete function $p(r_k) = \frac{n_k}{n}$, where r_k is the k^{th} gray level, n_k is the number of pixels in the image with that gray level, n is the total number of pixels in the image, and $k = 0, 1, 2, \ldots, L-1$. Loosely speaking, $p(r_k)$ gives an estimate of the probability of occurrence of gray-level r_k [4], where $\sum p(r_k) = 1$.

We have found zones in the histograms consisting of narrow bands (sometimes with a width of just one colour) clearly separated from each other. This is of interest to us because the automaton self-organizes through very specific sets of predominant cell values, or colours. These sets of colours vary depending on the settings of the of the transition rules parameters. By examining the evolution of these narrow bands, we surprisingly found that their envelopes (i.e., time trajectories) resemble the amplitude envelopes of the partials in the spectrum of sounds. With this in mind, we devised a method to generate sound spectrograms from CA histograms. Considering that the time domain is common for both the histograms sequences and the spectrogram, we map the histogram's sample space domain onto the spectrogram's frequency domain and, the histogram's probability domain onto the spectral magnitude domain.

The spectrograms are synthesised using a combination of additive synthesis and Frequency Modulation (FM) techniques [5]. Firstly, we select and extract the structures of the histograms sequence corresponding to the most predominant bins. Then, these structures are converted into partials of a spectrogram. We extract the amplitude envelopes and, without any kind of amplitude transformations, apply interpolation in order to define the duration of the sound (60 time steps per second). The frequency assignment for the partials is arbitrary; in our experiments we have assigned random frequencies to each partial in order to obtain complex ratios between them. By designing a bounded random frequency generator it is possible to obtain a wide range of different sounds. In order to render the partials we consider narrow bands of just one *colour width*. Then, we add frequency fluctuations by using FM. The FM is driven by the amplitude envelopes of each partial as follows: $f_p(t) = f_p + width * AmpEnv_p(t)$ where p denotes each of the partials.

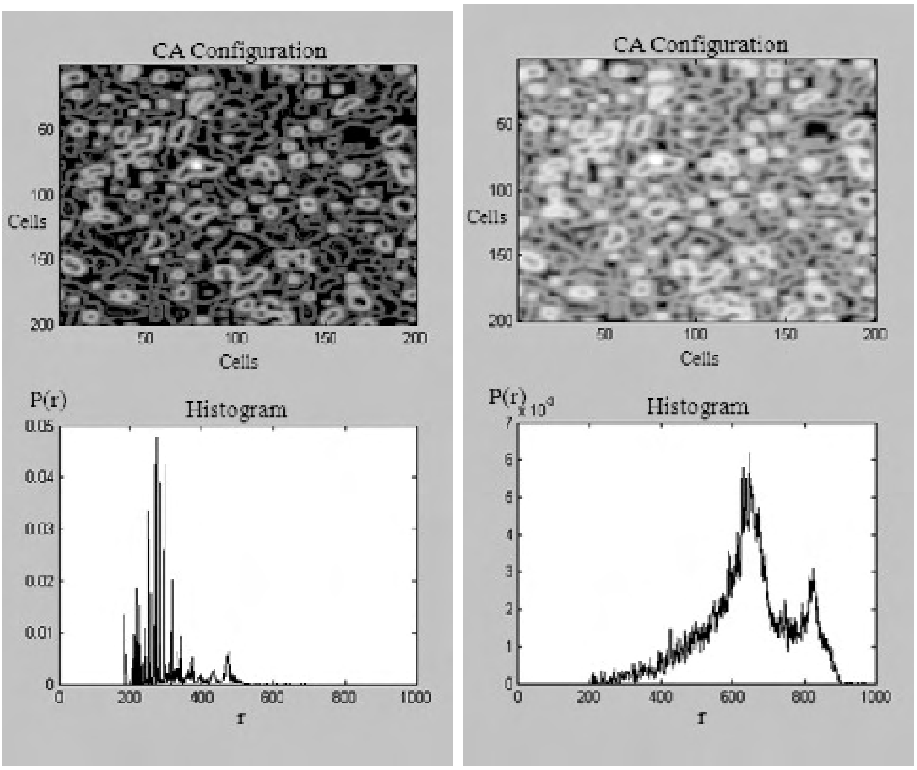
3 Making sense of the CA behaviour

We have studied the behaviour of the automaton under a variety of different settings in order to gain a better understanding of the histograms, with a view on predicting the types of sounds that different CA behaviours would produce.

3.1 Quasi-synchronic behaviour

We have discovered that quasi-synchronic CA behaviour (i.e., where all cells of the CA grid reach their maximum allowed value almost simultaneously) generates histograms that are very suitable for sound synthesis. Just after the cells reach their maximum value, patterns of *distorted circumferences* emerge. The contours of these patterns create narrow bands, or peaks, in the histogram. From here on, the cells values increase towards the maximum value and the boundaries of the distorted circumferences become less defined, creating wide bands in the histogram (Fig. 1). At each cycle of the automaton, this process is repeated but with slightly different distorted circumferences shapes, creating structures in the histogram with time varying amplitudes.

Figure 2 shows the frontal plot of a typical histograms sequence produced by a quasi-synchronic CA run. It also shows how the CA rules and parameter values are reflected in the histogram. We can see the two feedbacks, one positive (due to Rules 1 and 2), and one negative (due to Rule 3). Peaks E1 and E2 are the two CA end points corresponding to the minimum and maximum cell values. The probabilities of these values (or colours) always reach high values. This is



(a) (b)

Fig. 1. Different CA configurations in the ‘quasi-synchronic’ behavior and the histograms they produce: narrow bands (a) and wide bands (b).

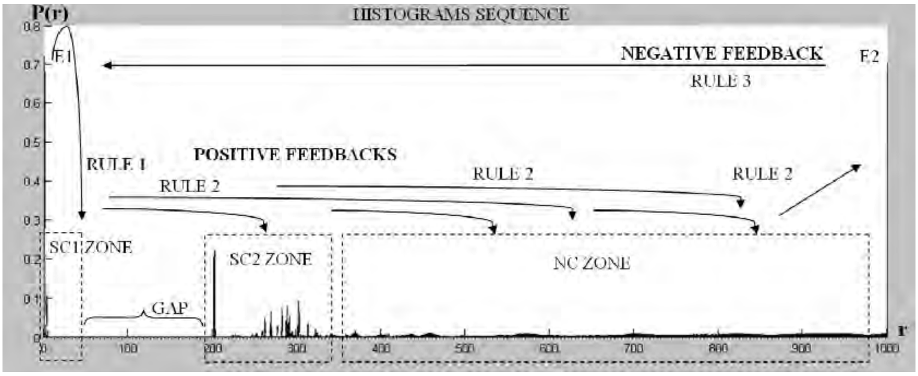


Fig. 2. Frontal plot of the histograms sequence of a CA evolution and the role of the CA rules.

so for E1, due to Rule 3, and for E2 it is due to the need to impose a limit on (or *cap*) the maximum cell value (e.g., when the result of Rule 2 is a value that is over the maximum allowed value). SC1 is a zone of narrow bands that appear due to Rule 1. Next to SC1 there is a GAP. This is a region that always has zero probability values, meaning that values corresponding to this area are ever reached by the automaton. The beginning of the GAP is due to Rule 1, which has a maximum possible value depending of r_1 and r_2; this limits the size of the SC1 Zone. The GAP and its size are due to the constant K in Rule 2, which is like an offset. SC2 is a zone of narrow bands due to Rule 2 applied to cells with values (or colours) in the range of SC1 and SC2 itself. Finally, in NC Zone there are wide bands that appear due to Rule 2.

The behaviour described above is granted by a variety of different combinations of CA parameter settings. We found interesting time evolving structures by working with large ranges of values, from hundreds and usually thousands. The images produced by the automaton with thousands of different values (or colours) proved to be more suitable for our purposes because they seem to be much more lively and natural than with just a few values. Typical parameters settings (for someone wishing to replicate our experiments) are: CA grid of 200 by 200 cells, K equal to a value between 20% and 30% of V, and both r_1 and r_2 set equal to 2.

3.2 Long-term behaviour

The effect of the automaton's long-term behaviour can be of two types: long-term behaviours that produce structures for sustained sounds and long-term behaviours that produce structures for non-sustained sounds. The previously mentioned quasi-synchronic behaviour is an example of the former type. CA definitions with fewer neighbours are likely to generate structures suitable for non-sustained sounds. This is the case when considering a Neumann neighbourhood [3] or even fewer neighbours; e.g., the case where the central cell is not a neighbour. For instance, if we consider fewer neighbours, the divisor of Rule 2 is lower than if we consider Moore neighbourhoods, and therefore infected cells will have greater chances of getting immediately ill. Ill cells in the neighbourhood cause a higher numerator while the denominator does not grow due to ill or healthy cells. This causes the automaton to reach a long-term behaviour with cells whose value oscillates between 0, cell values corresponding to SC1 Zone, and $V-1$. While reaching this long-term behaviour, the rest of the histogram's bins fade out to zero, creating the effect of sound release. These structures are interesting because different release times emerge for different histogram bins (Figs. 3 and 4). This kind of effect occurs in the sounds produced by most acoustic instruments.

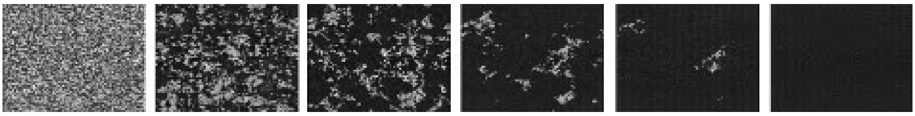


Fig. 3. CA behaviour that produces non-sustained sounds.

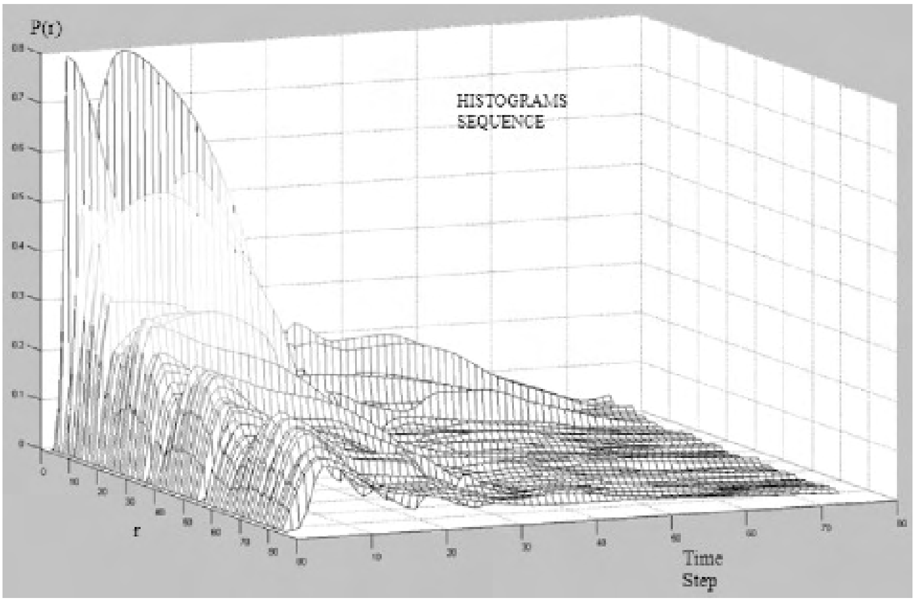


Fig. 4. Histograms sequence from a non-sustained structure obtained with a method for extracting smooth amplitude envelopes.

3.3 Spiral waves

A type of behaviour referred to as spiral waves create types of histograms that are of great interest for sound synthesis. Starting from an initial random configuration of cell values, the automaton often produces behaviour that resembles the quasi-synchronic behaviour mentioned earlier. But then, spiral waves start to develop, creating structures that resemble sound partials with increasingly high amplitudes. Once the spirals have completely evolved, they often expand themselves, covering the whole CA grid. This creates sequences of images, which are cyclic and stable. When rendered to sound, the amplitudes of the partials often stop increasing and settle to relatively stable values. What is interesting here is that the relative amplitudes of the partials are very similar to each other but with small differences in their time evolution (or envelopes) (Fig. 5). This is a general property found in most sounds produced by acoustic instruments. For instance, a sound played on a violin starts noisy due to the attack of the bow and then it settles into a periodic vibration. (When the violin is played by experienced violinists, we often cannot hear this noisy attack because the strings settle into vibration very quickly.)

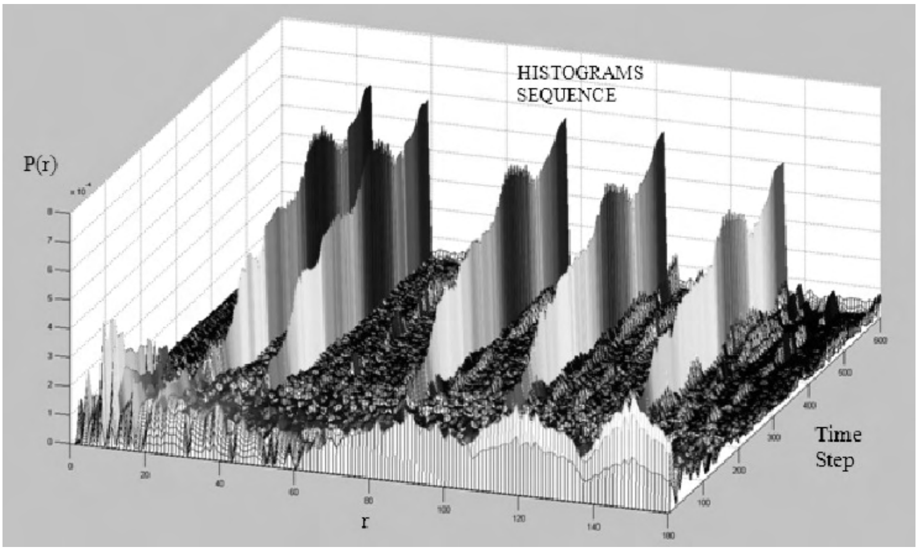


Fig. 5. Histograms sequence from spiral waves.

3.4 Time dynamics

Different CA behaviours can produce different types of spectral structures with different time evolutions. Time varying amplitudes can be considered in various ways. The original amplitude evolution of the histogram's bins usually display oscillatory behaviour, and it could be desired to perform an amplitude envelope extraction algorithm to get a smoother amplitude envelope (as the one shown in Fig. 4). Figure 6 shows the time evolution of one bin of a histogram corresponding to a quasi-synchronic behaviour (solid line). Note the prominent peaks due to the quasi-synchronic behaviour. The distance of the peaks is the period of the CA cycle. The dotted line indicates the amplitude envelope that could be extracted from this histogram.

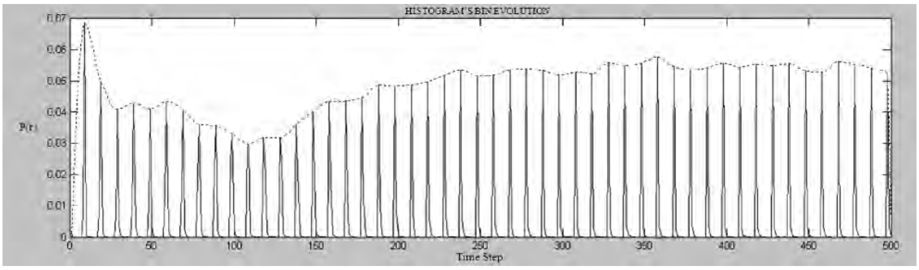


Fig. 6. Time evolution of one histogram's bin in the quasi-synchronic behavior (solid line), and its respective amplitude envelope (dotted line).

The self-organizing process of the automaton also produces interesting noisy structures. This is important for our research because in addition to the sinusoidal partials, noise is an important component of sounds, particularly at their onset (or *attack*) [5]. In Figs. 7 and 8 we can see a noisy structure at the beginning of the histograms sequence, which then disappears while more stable structures emerge.

When histogram structures have narrow bands, the respective spectra will contain deviations of partials frequency trajectories, which is another property commonly found in sounds produced by acoustic instruments (Fig. 8).

3.5 Invariance property

We have found an invariance property in the histograms sequences by studying different runs of the automaton (with enough time to reach the long-term behaviour) with the same settings, but starting from different initial uniform random configurations of cells values. By looking at a zone of narrow bands (peaks of prominent colours like SC1or SC2 in the quasi-synchronic behaviour) we have observed that the structures of the histograms in terms of peak locations remained identical for all cells; that is all cells ended up organized with the same set of prominent colours. The relative amplitudes of the peaks remained similar, but the time variations of the amplitudes (or envelopes) were slightly different for every run. This invariance property remains largely present even for different sizes of the CA grid (Fig. 9). It is therefore possible to automatically

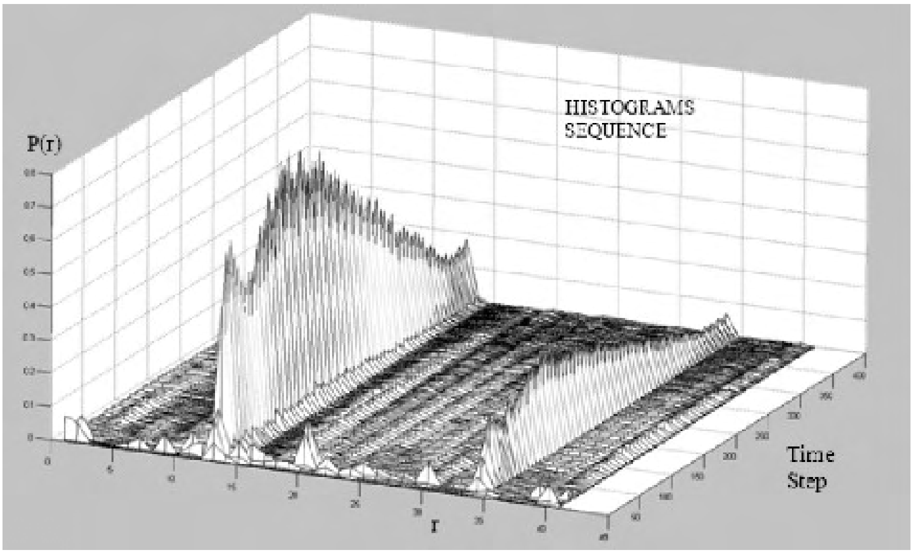


Fig. 7. Structures from SC2 Zone showing self-organization and correlated amplitudes.

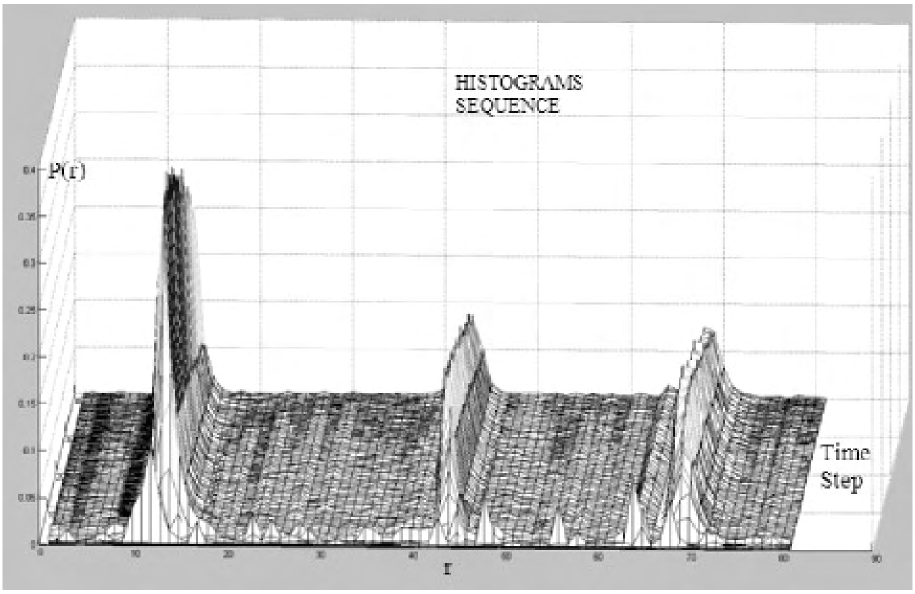


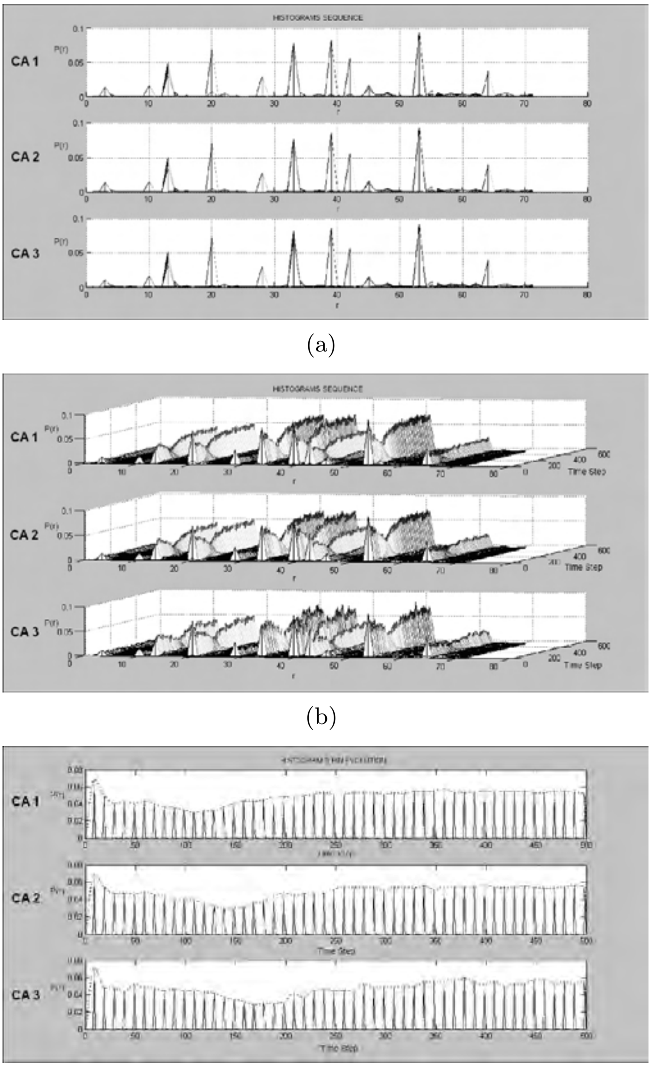
Fig. 8. Structures from SC2 Zone showing self-organization and deviation of structure trajectories.

obtain multiple instances of a certain type sound. All instances will share the same structure, but with differences in the time-varying amplitudes. Thus, we can design an instrument that would not output the same exact sound twice and therefore, capable of generating more natural sequences of notes.

Although we can expect, and we assume, that every run of the same automaton would display identical behaviour, it is not straightforward to ascertain in advance the value that a specific cell would hold after a number of generations. But it is possible to ascertain the position of the predominant peaks in the histogram. Thus, this is a considerable degree of predictability with applications for example in the context of our mapping. This does not mean that we will have this predictability when starting from any initial pattern of cells values (or colours). When starting from an image with a certain level of organization of colours, the resulting histogram would probably be different from when starting from a random distribution of colours; we have not tested such cases systematically yet.

4 Conclusion and ongoing work

The predictability of the outcome of CA evolution is an open problem [6]. Although a level of unpredictability is accepted, and often desired, in systems for generating music and sound, being under unpredictability conditions implies limited controllability. A lack of a reasonable level of control restricts the music or sound design process [7]. Our synthesis technique alleviates this limitation in many respects. As we have seen, the CA rules and parameters are very well reflected in the histogram. Thus, it is possible to find direct relations between the CA parameters values and their effects in the histogram. Most of them refer to the spectral dimension and some to the time dimension. For instance, the lower the value of K, the narrower is the GAP (Fig. 2). As a consequence, the quasi-synchronic behaviour produces more noise bands in the NC Zone. It is intuitive to see that a wide GAP implies less noise bands. The parameter K also contributes to the nature of the beginning of a sound; the lower the value of K, the longer it takes for the partials to appear in the histogram; in these cases, such delays are also proportional to the histogram bins. The parameters r_1 and r_2 control the tendency of a healthy cell to become infected by infected and by ill neighbours, respectively. With the quasi-synchronic behaviour, the lower the vakues of r_1 and r_2, the wider the SC1 Zone. If these values are extremely low, the histograms sequence will evolve presenting only two peaks in the first and the last bins. Conversely, if the values are higher than the number of neighbours, the histograms sequence will evolve, blocking the first bin. With respect to the time domain, we have seen that by considering fewer neighbours it is possible to control the time-evolution of the amplitude envelopes. Once analyzed how non-sustained structures appear, it is possible to induce the same effect working with Moore neighbourhoods, by modifying the rules. One way to work with fewer neighbours is by dividing the rule parameters A and B by two. Then one can obtain the same kind of non-sustained structures that is obtained when working with Neumann neighbourhoods.



(a)

(b)

(c)

Fig. 9. Invariance property in histograms sequence for three different CA runs: CA2 has the same definition than CA1 but starting from a different initial configuration (uniform random distribution). CA3 is the same definition than CA1 and CA2, but with different size, where the size of CA1 and CA2 are 200 by 200, and the size of CA3 is 100 by 100 cells). (a) Frontal plot. (b) Lateral plot. (c) Time evolution of one bin of the histograms sequence.

We are currently considering the possibility of using this technique to render sounds from real chemical reaction-diffusion processes.

References

[1] Doornbusch, P. (2005). *The Music of the CSIRAC: Australia's First Computer Music*. Victoria, Australia: Common Ground.

[2] Adamatzky, A., De Lacy Costello, B. and Asai, T. (2005). *Reaction-Diffusion Computers*. Elsevier.

[3] Gerhardt, M. and Schuster, H. (1989). A cellular automaton describing the formation of spatially ordered structures in chemical systems, *Physica D*, 36:209?-221.

[4] Pratt, W. K. (1978). *Digital Image Processing*. New York, NY: Wiley.

[5] Miranda, E. R. (2002). *Computer Sound Design: Synthesis Techniques and Programming*. Elsevier/Focal Press.

[6] Wolfram, S. (1984). Computational Theory of Cellular Automata, *Communications in Mathematical Physics*, 96:15–57.

[7] Miranda, E R. and Wanderley, M. M. (2006). *New Digital Musical Instruments: Control and Interaction beyond de Keyboard*. Middleton, WI: A-R Editions.

.

Mapping physical phenomena onto CA-models [⋆]

Olga Bandman

Supercomputer Software Department
ICM&MG, Siberian Branch, Russian Academy of Sciences
Pr. Lavrentieva, 6, Novosibirsk, 630090, Russia
bandman@ssd.sscc.ru

Abstract. Simulation of natural phenomena using CA-models attracts more and more attention of researchers and enters into practical use. However, the methodology of constructing CA-models for simulating certain natural process as well as analyzing simulation results, are not yet completely developed. The reason lays in a high level of CA-models abstraction which causes difficulties in establishing correspondence between CA parameters and physical values characterizing the process under simulation. In the paper an attempt is made to formulate general principles of constructing CA simulation models, and to show in detail their application to three types of CAs: diffusion CAs, Lattice-Gas CAs, and asynchronous probabilistic CAs. In all cases algorithms for scaling coefficients are developed and illustrated by Examples.

1 Introduction

Cellular Automata (CA) is nowadays an object of growing interest as a mathematical model for spatial dynamics simulation. Due to its capability to simulate nonlinear and discontinuous processes, CAs are expected [1, 2] to become a complement to partial differential equations (PDE), especially in case when there is no other mathematical model of a phenomenon which is to be investigated. By now, a great variety of CAs are known whose evolution simulates certain kinds of spatial dynamics in physics, chemistry, biology, as well as a behavior of colonies of animals or crowds of peoples.

The most known and advanced CA-models may be divided into three classes: (1) CA-models of processes containing diffusion in explicit form, simulating diffusion-reaction processes [3, 4, 5, 6], (2) Lattice-Gas CAs simulating waves and fluids [7, 8, 9], and (3) asynchronous probabilistic CA-models simulating molecular kinetics [10, 11, 12, 13, 14, 15]. Although the above CA-models are well studied, there is no systematic approach to establish relations between real values characterizing the process to be simulated and the parameters of its CA-model. Meanwhile, the abstract nature of CA-models causes sometimes significant difficulties in constructing an appropriate CA, i.e. in determining the size of the CA,

[⋆] Supported by Russian Academy of Sciences, Basic Research Program of N 14-16(2007)

expressing initial and border conditions in CA terms, defining transition rules, probabilities of their application and so on. The inverse problem arises when transforming the obtained CA simulation results into the real physical values.

To make the situation a bit more clear, a general approach to put into correspondence a real process and its CA-model is proposed. Based on it the techniques for determining scaling coefficients for three above classes of CA-models are presented.

Apart from Introduction and Conclusion the paper contains four sections. In Sect. 2 some formal definitions are given and general principles for CA-models construction are formulated. The next three sections are devoted to detailed considerations of obtaining scaling coefficients relating physical phenomena and its CA-model for the three classes of CAs.

2 General principles of CA-models construction

A CA simulation problem is usually stated by defining the following data:

- Size and form of the domain under simulation.
- Properties of the medium where the simulated process is going on.
- Initial and border conditions.

The results to be obtained are scalar $u(\mathbf{x}, t)$ or vector $\mathbf{u}(\mathbf{x}, t)$ functions, which may represent spatial dynamics of certain physical phenomenon. To achieve simulation goal a CA-model should be properly constructed, which may be done according to the following principles.

- The CA should be valid for the whole domain of values of the phenomenon to be simulated. For example, when simulating the fluid flow using Lattice-Gas model it is necessary to be sure that the Reynold's number does not exceed the Lattice-Gas CA limits.
- Implementation of CA simulation should be feasible on available computer systems during the acceptable time.
- It is important to chose the set of basic parameters of a CA-model, whose scales are straightforward. Usually, they are invariant of the phenomenon to be simulated. Such invariants are Reynold's number in fluid dynamics, dimensionless diffusion coefficient, reaction rates. Based on them all other scales may be derived.

It is worth to notice, that the choice of scaling parameters in CA-simulation is similar to that in PDE solution in mathematical physics. The main difference is in the fact that in numerical methods only two scaling values are needed: time step h and space step τ. All other computed values need not to be transformed, except in the special methods.

As for CA-modeling, the choice of CA parameters is more complicated, which is caused by the following factors.

1) The scales should be chosen for all the values involved in the simulation process, including the medium properties such that density, viscosity, pressure, sound velocity, etc.

2) CA-simulation application experience is not rich enough for obtaining quantitative values of above properties in CA-model terms. So, sometimes, they should be obtained by experiments on typical, a priory studied, processes, which is both time consuming and not very accurate.

3) CA-models are very diversified, most of them are essentially nonlinear with nonpredictable behavior, exhibiting all features of complexity [16], and, hence, all problems associated with it.

In what follows the measure system MKS (meter, kg (mass), sec) is used for physical values. As for CA-model, their parameters are mostly dimensionless, expressed in numbers of cells, whose side length $l_{CA} = 1$, number of particles Np_{CA} with mass $m = 1$, sum of velocity vectors with $|\mathbf{v}_{CA}| = 1$ and so on. Scaling coefficients (*scales* for short) are defined as relations of the physical value to its CA counterpart and denoted by $\mu_z = z/z_{CA}$, z being any value involved in the model.

Scales are divided into three groups.

The first group includes two *fundamental scales* which are defined in all types of numerical models, they are time step $\mu_l = h$ and space step $\mu_t = \tau$. In CA-models they represent the linear size of a cell and the time elapsing between two sequent global states, respectively.

The second group comprises *medium properties scales*, such as viscosity, sound speed, substance density, which characterize the medium where the process proceeds, but are not the simulated values. The CA properties of this group are the characteristics of the model derived from CA transition rules or, if it is not possible, they are obtained by simulation experiments.

The third group comprises the *scales of simulated values,* i.e. the values of functions $u(t, \mathbf{x})$, which are the objectives of the simulation, such as the velocity of a fluid flow or an acoustic wave, the rate of a crystal growth or of quantity of a substance obtained by a chemical reaction, etc.

Scales from the second and the third groups are strongly dependent of the nature of the phenomenon under simulation. Hence special methods for any process are further needed considered each in its own section.

A formalism further used for representing CA-models is Parallel Substitution Algorithm [17]. According to it a CA-model $\aleph$ is represented by four notions $\aleph = \langle A, M, \Theta, \varrho \rangle$, where A is a set of symbols of any kind, $M = \{m_1, m_2, ...\}$ is an enumerable set, Θ is a set of local operators, and ϱ is a mode of operation, which determines the time-space distribution of operator application. The central concept in the CA-model is a *cell,* which is a pair (a, m), where $a \in A$ is a cell state, and $m \in M$. The set of cells $\Omega = \{(a_i, m_i) : i = 1, ...\}$, containing no cells with identical names is called a *cellular array.*

On the naming set M *naming functions* $\varphi(m)$ are defined which whose values indicate the location of cells communicating with a cell named m. When Cartesian coordinates $M = \{(i, j)\}$ are used for names, the naming functions are usually given in the form of shifts $\phi_k = (i + a, j + b)$, a, b being integers. A set of naming functions

$$T(m) = \{m, \phi_1(m), \ldots, \phi_n(m)\}, \quad n \ll |M|, \tag{1}$$

is referred to as a *template*, m being called as *active cell* of a template.

A subset of cells

$$S(m) = \{(u_0, m), (u_1, \phi_1(m)), \ldots, (u_n, \phi_n(m))\}, \tag{2}$$

having the names from $T(m)$, is called a *local configuration* with $T(m)$ as its *underlying template.*

A local operator $\Theta_i \in \Theta$ is expressed in the form of a *substitution* [17] of local configurations as follows

$$\Theta(m) : S(m) \star S''(m) \rightarrow S'(m), \quad \forall m \in M, \tag{3}$$

the underlying templates of $S(m)$ and $S'(m)$ being identical, i.e. $T'(m) = T(m)$, and that of $S''(m)$ $T''(m)$ being allowed to be arbitrary.

An application of $\Theta(m)$ to a certain cell $(u, m) \in \Omega$ (a single-shot application) consists in removing the cells of $S(m)$ from Ω and replacing them by the cells given in $S'(m)$. Such a concept of a local operator allows to simulate living organisms which may grow and die. When simulating physical phenomena *stationary* local operators [17] are used which do not change the naming set, only replacing the states of cells from $S(m)$ in (2) by the states of cells from

$$S'(m) = \{(u'_0, m), (u'_1, \phi_1(m)), \ldots, (u'_h, \phi_h(m))\}$$

u' being obtained according to transition functions

$$u'_k = f_k(v_0, v_1, \ldots, v_n, v_{n+1}, \ldots v_{n+h}), \tag{4}$$

where the last h variables are the states of cells from $S''(m)$. The latter in not changed by application of Θ to the cell named m, but provides f_k with additional variables, playing a role of a context [17].

There are different modes of local operator application ordering to perform a global transition $\Omega(t) \rightarrow \Omega(t+1)$. *Synchronous mode* provides for transition functions (4) to be computed using the current cell-states, i.e. $S(m) \subset \Omega(t)$. It may be performed at once (in parallel) or in any order. *Asynchronous mode* of operation suggests the substitution of cell states in $S(m)$ being done immediately after $\Theta(m)$ is applied. So, $S(m) \subset \Omega(t) \cup \Omega(t+1)$. In both case the transition to the next global state is referred to as an *iteration* occurring when all cells have completed their computations. The sequence

$$\Omega(0), \Omega(1), \ldots, \Omega(t), \ldots, \Omega(\hat{t})$$

is called a *CA evolution*, $\hat{t}$ denotes a terminal iteration number.

Performing a CA simulation task comprises three stages:

1) constructing the model, i.e. determining the CA $\aleph = \langle A, M, \Theta, \varrho \rangle$ and and its initial global state $\Omega(0)$,

2) obtaining resulting data by running the CA program , and

3) interpreting the results by transferring the model parameters into habitual physical values.

The first and the third stages require scaling coefficients to be known. The problem is solved differently for different types of CA-models, but the techniques rely on the same above principles and the same formalism.

3 CA-models of phenomena which include diffusion

Diffusion components are present in the majority of natural spatially distributed processes, in PDEs being represented by a Laplacian

$$d(u_{xx} + u_{yy}) \tag{5}$$

where d - is a diffusion coefficient in $\mathtt{m}^2\mathtt{s}^{-1}$. Being discretized by means of the “cross template" (5) takes the form

$$C(u_{i-1,j} + u_{i,j+1} + u_{i+1,j} + u_{i,j-1} - 4u_{i,j}), \quad i = x/h,\ j = y/h, \tag{6}$$

with

$$C = \frac{d\tau}{h^2}, \tag{7}$$

where h and τ are length and time steps, respectively. In other words, they are length and time scales, denoted further as μ_l and μ_t. In computational mathematics they are chosen according to performance constraints, convergence and stability conditions, and, perhaps some special requirements.

As for CA diffusion, determination of scales is more complicated and less studied. The reason is in the fact that each CA-model is characterized by its own CA-diffusion coefficient C_{CA}, which may be obtained analytically, as it is done in [4], or experimentally [3]. The coefficient is dimensionless and plays the same role that C in (6), being an invariant of a diffusion model. So, when a CA-model is chosen C_{CA} is easily obtained.

Taking into consideration (7) the derivation of a diffusion CA-model scaling coefficients is based on the following relation.

$$C_{CA} = d\frac{\mu_t}{\mu_l^2}, \tag{8}$$

Since in any certain simulation task d is a known characterizing property of the medium and the size of the area under simulation $L_x \times L_y \times L_z$ is given, it is sufficient to chose the CA dimensions $N_i \times N_j \times N_k$ for obtaining all scales. Usually CA size is determined based on the required resolution of resulting spatial function and the available computing resources. All scaling coefficients are straightforward.

$$\mu_l = \frac{L}{N}\ \mathtt{m}, \quad \mu_d = \frac{d}{C_{CA}}\ \mathtt{m}^2\mathtt{s}^{-1}, \quad \mu_t = \frac{\mu_l^2}{\mu_d}\ \mathtt{s}. \tag{9}$$

There are several CA diffusion models. Two of them are the most popular. The first is a *two-stage synchronous* probabilistic model (CA-synch), proposed in [1] and studied and founded in [4]. The second, called in [1] a *naive diffusion* (CA-naive), is an asynchronous probabilistic CA . For all models C_{CA} is a function of the probability used in the local operators, the maximal values $\hat{C}_{CA}$ being fixed for each diffusion CA.

In Table 1 $\hat{C}_{CA}$ for the above two CA-diffusion models are given for 1D, 2D, and 3D cases. For $C_{CA-synch}$ they are proved in [4], for $C_{CA-naive}$ they are obtained by comparing simulation results with analytically known values.

Table 1. Maximum values of diffusion coefficients of synchronous and asynchronous CA-models

n	$\hat{C}_{CA-synch}$	$\hat{C}_{CA-naive}$
1D	1.0	1.0
2D	3/2	1/2
3D	23/18	1/3

By varying probabilities of local operators application it is possible to simulate processes with diffusion coefficient changing in space or in time, for example, depending on temperature.

Example 1. *Application a diffusion-convection CA-model to simulate vapor propagation through a porous membrane.* The objective of simulation is to investigate the impact of pore walls properties on the percolation velocity. A fragment of the membrane in a 2D approximation is considered, the pore diameter being equal to its width. The fragment has three vertically located pores with equal diameters: a hydrophilic, a hydrophobic and a neutral one (Fig. 1). The process is represented as a stream of abstract particles, moving under convective and diffusive forces. The source of particles is on the fragment top area where the pressure of the vapor is imposed having equal effect to the three pores. The convective force makes the vapor particles move downwards. The diffusion component accounts for anisotropic properties of the medium by varying probability values of particles displacements along the pore diameter, depending on pore wall properties. The mode of percolation is characterized by the relation of the impact of convective component to that of the diffusion one, defined as $Pe = 0.5$ [18]. The porous sample to be investigated has the size $L_i = L_j = L = 0.03$ m, pore diameter being $Dp = 8 \cdot 10^{-3}$ m. Diffusion coefficient of vapor is $d = 10^{-4}\ \mathtt{m}^2\mathtt{s}^{-1}$, the density is $\rho = 0.5 \cdot 10^3\ \mathtt{kg} \cdot \mathtt{m}^{-3}$.

The model to be used is an asynchronous naive CA-diffusion with $A = \{0, 1\}$ and the naming set $M = \{(i, j) : i, j = 0, \dots, N-1\}$. The local operator is the superposition of two substitutions: $\Theta = \{\Theta_{conv}, \Theta_{diff}\}$, both being probabilistic. The probabilities p_{conv} and p_{diff} are determined according to the given coefficient $Pe = p_{conv}/p_{diff} = 0.5$ [18],

$$\Theta_{conv}(i,j) = \{(1,(i,j))(0,(i+1,j))\} \xrightarrow{p_{conv}} \{(0,(i,j))(1,(i+1,j))\}, \qquad (10)$$

advances a particle along the pore with a probability equal to p_{conv}.

$$\Theta_{diff}(i,j) = \{(1,(i,j))(0,(i+a,j+b))\} \xrightarrow{p'_{diff}} \{(0,(i,j))(1,(i+a,j+b))\}, \quad (11)$$

exchanges the cell-state $u = 1$ with one of its neighbor according to the pair $(a, b) : a, b \in \{-1, 1\}$, which is chosen with the probability $p = 1/q$, q being the number of empty neighbors. So, $p'_{diff} = p_{diff} \cdot p_w$, with p_w accounting for pore type and depending of the distance g_w between the cell (i, j) and the wall. When a pore is neutral or if $g_w > \beta$, then $p_w = 1$, β being the distance where wall influence may be neglected. If $g_w \leq \beta$, then for hydrophilous and hydrophphobic pores

$$p_w(\texttt{phil}) = 1 - \exp(-g_w/n_1\beta_1), \quad p_w(\texttt{phob}) = \exp(-g_w/n_2\beta),$$

respectively, n_1, n_2 depending of the wall properties.

The above data are sufficient for obtaining physical scales of CA-model parameters.

1) *Length scale.* According to porous medium simulation practice [18] minimal diameter of the pore should be several times larger than 10. Taking $h = \mu_l = 10^{-4}$ the CA-diameter yields $Dp_{CA} = 80$, and the linear size of the whole CA $L_{CA} = 300$.

2) *Diffusion scale.* Since the 2D naive asynchronous CA diffusion is used, $C_{CA} = 0.5$, and $\mu_C = d/C_{CA} = 2 \cdot 10^{-4}$ $\texttt{m}^2\texttt{s}^{-1}$.

3) *Time scale* $\mu_t = \mu_l^2/\mu_C = 0.5 \cdot 10^{-4}$ s.

4) *Density scale* (mass of an abstract particle) $\mu_\rho = \rho/(\rho_0/h^3) = 1.25 \cdot 10^{-9}$ kg.

5) *Flow scale* (mass of vapor propagating through the cross-section of a pore per second, relative to the number of particles passing through the pore per iteration). Accounting for a pore square $Sp = 5024$ cells, $\mu_Q = \mu_\rho/\mu_t = 2 \cdot 10^{-5}$ kgs^{-1}.

CA-simulation results in the following flows, obtained in particles numbers passing through a pore per iteration

$$Q_{CA}(\texttt{phob}) = 3939, \quad Q_{CA}(\texttt{phil}) = 1975, \quad Q_{CA}(\texttt{neutr}) = 2812,$$

which are recalculated into physical values by multiplying the obtained Q_{CA} by μ_Q resulting in the following flows.

$$Q(\texttt{phob}) = 0.079 \ \texttt{kgs}^{-1}, \ Q(\texttt{phil}) = 0.039 \ \texttt{kgs}^{-1}, \ Q(\texttt{neutr}) = 0.058 \ \texttt{kgs}^{-1}.$$

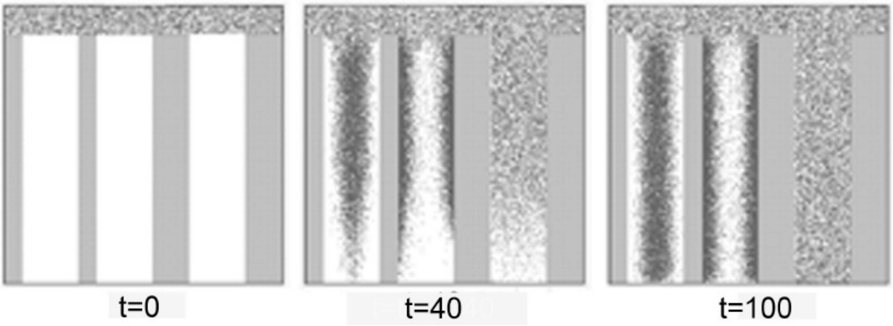


Fig. 1. Three snapshots of CA simulation of vapor propagation through porous membrane.

4 Lattice-Gas models of viscous flows

Gas-Lattice CA-models comprise a special class of CA intended to simulate processes in gas and liquids. The medium is represented by abstract particles, moving in a discrete space and colliding. The most known and well studied model is a series of stochastic Lattice-Gas CAs called FHP-models according to the names if the authors [9]. Formally, they are synchronous CA, characterized by the following parameters. The naming set $M = \{m_k : k = 1, 2, \ldots, |M|\}$ enumerates hexagons on a 2D plane. A cell neighborhood includes the cell names of 6 adjacent cells. Accordingly, 6 moving and some rest particle may be located in a cell. To represent the cell states with 6 moving and one rest particle the alphabet $A = \{s = (s_0, \ldots, s_6), |A| = 2^7$ comprises Boolean vectors 7 bit long. A component of a state vector $s_i = 1$ indicates that the cell (s, m) has a particle moving towards the ith neighbor $(i = 1, \ldots, 6)$ with a velocity $c_i = 1$, or if the cell has a rest particle, then $s_0 = 1$ having the velocity $c_0 = 0$. Particle mass is equal to 1.

Two local operators determine the functioning of a CA. The first Θ_1 makes all particles in all cells simultaneously propagate one cell towards the neighbor pointed by its velocity vector. It is convenient to represent it as a set of 6 substitutions each being applied to ith component of state vector.

$$\Theta_1(m, i) = \{(s_i, m)\} \star \{(s'_i, \varphi_i(m))\} \to \{(s'_i, m))\}, \quad i = 1, 2 \ldots, 6. \tag{12}$$

The second contextless local operator simulates the collision of particles.

$$\Theta_2(m) = \{(s, m)\} \to \{(s\prime, m))\}. \tag{13}$$

The transition function $s\prime = f(s)$ is given in the form of a table, some arguments having two equiprobable outcomes. The principles of collision rules functioning is shown in Fig. 2.

The mode of operation of Lattice-Gas CA is two-stage synchronous, i.e. each iteration consists of two stages: on the first stage the six propagation operators

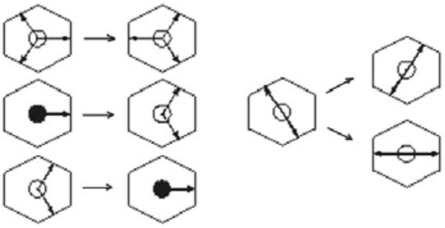


Fig. 2. Graphical representation of collision operators in FHP-I Lattice-Gas model. Deterministic rules are given on the left, the probabilistic ones — on the right

(12) act simultaneously, on the second stage the collision operator completes the transition to the next global state. In [9] FHP-model is proved to be identical to Navier-Stokes equation in describing the velocity of viscous flow and to be identical to the wave propagation when being regarded relative to particles density.

The 3D-version of the FHP-model called RD-1 is also known. It is based on the discrete space with cells having the form of rhombododecahedron. The template of RD-1 has 12 neighbors. It allows to simulate flows in large enough space, as compared with the FCHC model proposed and investigated in [9]. Although the model meets the isotropic conditions approximately, its experimental testing showed acceptable plausibility to the phenomenon [19].

Each Lattice-Gas CA-model is characterized by by the following parameters: equilibrium density $\tilde{\rho}_{CA}$, sound velocity Cs_{CA} and and viscosity $\nu_{CA}(\rho_{CA})$, the latter being a functions of particle density. In (Table 2) these parameters are given according to [9, 19].

Table 2. Main parameters of Lattice-Gas CA $\tilde{\rho}_{CA}$

CA-model	Cs_{CA}	$\tilde{\rho}_{CA}$	$\nu_{CA}(\tilde{\rho}_{CA})$ [1]
FHP	$1/\sqrt{2}$	1.2	0.85
RD-1	$\sqrt{7/13}$	3.9	0.325

In its turn a simulation task statement includes the following data in physical terms.

1) Size of the reservoir where the simulated process proceeds, l m being its characteristic length.

2) Location of source and of the outlets of the fluid.

[1] CA-viscosity values are given already corrected with regard to the isotropy coefficient $g(\rho)$ [9, 19].

3) The fluid properties: density ρ $\mathtt{kgm}^{-3}$, kinematic viscosity ν $\mathtt{m}^2\mathtt{s}^{-1}$.

3) External and initial conditions: working pressure P or pressure drop $\varDelta P$ $\mathtt{kgm}^2$.

4) The wanted velocity value u or the Reynolds number

$$Re = \frac{u \cdot l}{\nu}. \tag{14}$$

The above CA parameters and given physical values allow to obtain all scaling coefficients. Namely, the viscosity and the density scales are straightforward.

$$\mu_\nu = \frac{\nu}{\nu_{CA}}\ \mathtt{m}^2\mathtt{s}^{-1}, \quad \mu_\rho = \frac{\rho}{\tilde{\rho}_{CA}}\ \mathtt{kgm}^3. \tag{15}$$

Moreover, from the limiting CA Reynold's number [9]

$$Re = Re_{CA} = Cs_{CA}\frac{l_{CA}}{\nu_{CA}}$$

it is possible to find the characteristic CA-values of length l_{CA} or velocity u_{CA}, depending on the simulation task objective, and then calculate the following scales.

$$\mu_l = \frac{l}{L_{CA}}\ \mathtt{m}, \quad \mu_u = \frac{\mu_\nu}{\mu_l}\ \mathtt{ms}^{-1}, \quad \mu_t = \frac{\mu_l}{\mu_u}\ \mathtt{s}. \tag{16}$$

The pressure scale may be obtained in two ways:

$$\mu_P = \frac{\mu_u^2}{\mu_\rho}\ \mathtt{kgm}^{-2}, \quad \text{or} \quad \mu_P = \frac{P}{\rho_{CA}}\ \mathtt{kgm}^{-2}. \tag{17}$$

Since the alphabet, local operators and operation mode are defined by the Lattice-Gas model, only the linear dimensions and external pressure are to be determined in order to construct a Lattice-Gas CA-model.

1) The linear dimensions l_{CA} along the three directions l_i, l_j, l_k are computed by dividing the given real dimensions of the reservoir l_x, l_y, l_z by μ_l.

2) The external pressure is represented by the value of ρ_{CA} at the source location $\rho_{CA} = \frac{P}{\mu_P}$.

Example 2. *The Lattice-Gas 3D model RD-1 is used for simulating the flow of motor oil through a tube partly barred up by a choker.* The follwing data are given.

1) Tube diameter $Dt = 0.7$ $\mathtt{m}$, the length l= 7 $\mathtt{m}$, on one end of the tube the pressure $P = 1.07$ $\mathtt{atm}$ is imposed, the other end is left opened. So, pressure drop $\varDelta P = 7000$ $\mathtt{kgm}^{-1}$. The choker is placed at the distance 3.5 $\mathtt{m}$ from the source and barres up a half of the tube cross section.

2) Motor oil viscosity $\nu = 4.5 \cdot 10^{-3}$ $\mathtt{m}^2\mathtt{s}^{-1}$.

3) Motor oil density $\rho = 880$ $\mathtt{kgm}^{-3}$.

Simulation aims to obtain the velocity field behind the choker-bar.

Properties of the fluid being known the scales of density and viscosity are straightforward.

$$\mu_\nu = \frac{\nu}{\nu_{CA}} = 13.8 \cdot 10^{-3}\ \mathtt{m}^2\mathtt{s}^{-1}, \quad \mu_\rho = \frac{\rho}{\tilde{\rho}_{CA}} = 225\ \mathtt{kgm}^{-3}. \tag{18}$$

Taking the averaged CA-density $\rho_{CA}(0) = 8$ in source cells twice as large as the $\tilde{\rho}_{CA}$, the pressure scale is obtained according to (17). Then, accounting for (18) it is possible to calculate all others scales as follows.

$$\mu_P = \frac{\Delta P}{\rho_{CA}(0)} = 875\ \mathtt{kgm}^{-2}, \quad \mu_u = \sqrt{\frac{\mu_P}{\mu_\rho}} = 1.97\ \mathtt{ms}^{-1},$$

$$\mu_l = \frac{\mu_\nu}{\mu_u} = 7 \cdot 10^{-3}\ \mathtt{m}, \quad \mu_t = \frac{\mu_l}{\mu_u} = 3.55 \cdot 10^{-3}\ \mathtt{s}. \tag{19}$$

Having the scales in hands CA length values are computed as follows.

$$Dt_{CA} = \frac{Dt}{\mu_l} = 100, \qquad l_{CA} = \frac{l}{\mu_l} = 1000,$$

l_{CA} being the CA length of the tube (along the j-axis).

Simulation results are obtained as an 3D array of vectors $\langle \mathbf{u}_{CA}\rangle(i,j,k)$, each being equal to the averaged value of CA-velocity in a cell. The corresponding physical values

$$\mathbf{u}(x,y,z) = \mu_l \cdot \langle \mathbf{u}_{CA}\rangle(i,j,k), \quad x = i\mu_l, y = j\mu_l, z = \mu_l.$$

A fragment of the velocity field behind the choker is shown in Fig.3.

Maximum averaged value of the velocity over the choker is $\langle \mathbf{u}_{CA-max}\rangle = 0.9$ which yields $u_{max}1.77\ \mathtt{ms}^{-1}$. The mean velocity through the tube is computed as the average lengthwise velocity component at distance $l_{CA} = 700$ from the source, which occurred to be $\langle u_{CA}\rangle = 0.67$, which after scaling yield $\langle u\rangle = 1.3\mathtt{ms}^{-1}$.

5 Asynchronous CAs modeling nano-kinetic processes

Kinetic asynchronous CA (ACA) simulate phenomena consisting of sets of elementary actions, directly mimicking physical movements and interactions of molecules or atoms, in CA-models being referred to as called*particles*. Nowadays a number of surface nano-kinetic processes simulation by ACA are known, being, sometimes, referred to as Monte-Carlo methods [10, 11, 14, 20]. In brief, the methods are as follows. The domain where the process should proceed is divided into sites. A site (a cell in the CA) may be occupied by a particle representing a species from a given set of species involved in the process. The particles move and interact according to the laws prescribed by the phenomenon under simulation. The symbols, denoting different particles of the set comprise the ACA alphabet. The following alphabets are mostly used.

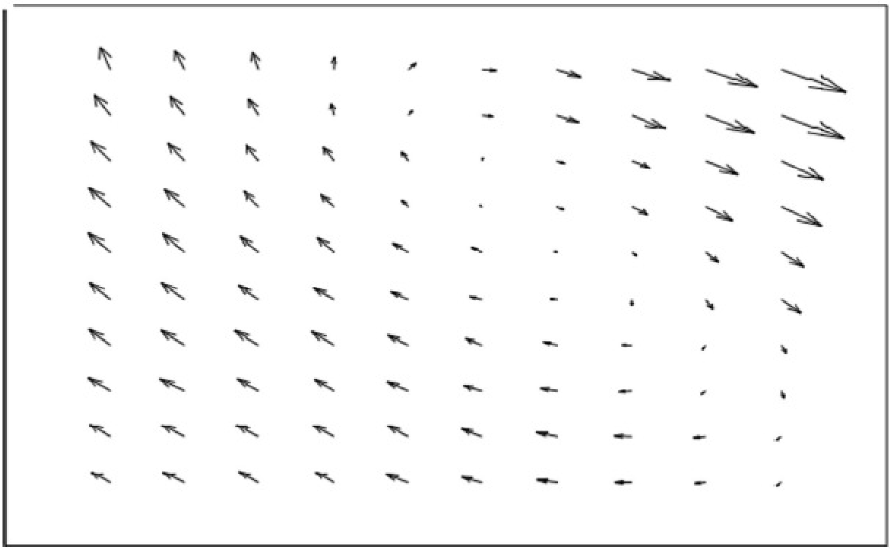


Fig. 3. A fragment of a lengthwise projection of of the velocity field behind the choker obtained by applying MathCad tools to RD-1 simulation results of the fluid flow through a tube

- Boolean alphabet $A = \{0, 1\}$, when it suffices to indicate presence or absence of a particle in the cell.
- A set of characters or symbols representing the notations of particles ($\emptyset$, Al, CO, H_2, etc.), a symbol $\emptyset$ denoting the state of unoccupied site.
- A set of integers, when several particles are allowed to be allocated in a single site.

The set of cell coordinates $M = \{(i,j) : i,j = 0,\ldots,N\}$ is frequently appended by a contextual domain consisting of a single generalized cell, say *Gas*, which represents the external space, where nothing happens, but wherefrom particles emanate and whereto the desorbed ones are gone. There is no restriction on the naming set structure, the most popular are arrays based on crystalline lattices, as well as those providing structural anisotropy.

Local operators which simulate the most used elementary actions in kinetic ACA are as follows. *Adsorption.* A particle $a \in A$ is adsorbed from the gas onto an empty site on a solid surface with the probability p_a.

$$\Theta_a : \{(\emptyset, m)\} \star \{(a, Gas)\} \xrightarrow{p_a} \{(a, m)\}. \tag{20}$$

Desorption. A particle b is desorbed from a surface site with the probability p_b.

$$\Theta_a : \{(a, m)\} \xrightarrow{p_b} \{(\emptyset, m)\}. \tag{21}$$

Reaction. If the particles a and b occur in the adjacent sites on the surface, they react forming a molecule ab, which outgoes to the gas with probability p_{ab}.

$$\Theta_{ab} : \{(a,m)(b,\varphi(m)) \overset{p_{ab}}{\rightarrow} \{(\emptyset,m)(\emptyset,\varphi(m))\}. \tag{22}$$

Diffusion. If a particle occur in the neighborhood of an empty site, it moves there with the probability p_d

$$\Theta_d : \{(a,m)(\emptyset,\varphi(m))\} \overset{p_d}{\rightarrow} \{(\emptyset,m)(a,\varphi(m))\}. \tag{23}$$

Of course, the above local operators do not exhaust all possible actions in surface kinetic processes, but they are typical and the most commonly encountered.

Nano-kinetic ACA-model simulation aims to investigate the behavior of the process both qualitatively (finding out whether it tends to a steady state, or to oscillation, or exhibits instability, etc.) and quantitatively (calculating amounts of obtained reactants, real rates of the reactions). The given data comprise a set of species, involved in the process, prerequisites to be accounted for, initial conditions. As for the size of the sample where the process should proceed, it is mostly constrained by the available computational resources rather than by researcher's request, the latter being usually “the larger the better".

From the above it is clear that local operators of a kinetic ACA model are very simple being easily derived from the chemical representation of the reaction mechanism. The problem is to determine the time scale and to find correct probabilities for reaction operators. The latter ones are computed according to chemical and physical properties of the process under simulation, usually calculated on the basis of molecular dynamics laws [21]. So, there are no general method to obtain their values. But, based on available experience, some following considerations may be made.

For a chemical reaction $R_i \in R$, $i = 1, \dots, n$, R being a set of reactions in the process, the probability in Θ_i is equal to

$$p_i = \frac{k_i}{\sum_{j=1}^{n} k_j},$$

where $k_i, k_j(\ \mathtt{s}^{-1})$, $j = 1, \dots, n$, are known rate coefficients.

Adsorption probability p_a is obtained apart for each type of particles according to given partial pressure of the corresponding species in the Gas and sticking coefficients. If any deposited atom is assumed to be adsorbed, then $p_a = 1$.

Diffusion probability p_d strongly depends on bond strength between particles and is computed based on bond energy values [21].

Time scale is calculated basing on the assumption that all time intervals Δt between a certain sequential event are identical, and, hence, may be put into correspondence with CA iterations Δt_{CA}, depending of what value is given. In [20] two methods of establishing a relation between Δt and Δt_{CA} are given: 1) when the real flow of atoms $F\ \mathtt{m}^{-2}\mathtt{s}^{-1}$ to be involved in the process per second is known, and 2) when the deposition rate (numbers of monolayers deposited onto the surface per second) $ML\ \mathtt{s}^{-1}$ is measured. Accordingly,

$$\mu_t = \frac{N_d \cdot t_{CA}}{\mu_l^2 \cdot F \cdot |M|}\ \mathtt{s}, \quad \text{or} \quad \mu_t = \frac{1}{ML \cdot |M|}\ \mathtt{s}. \tag{24}$$

where N_d is the total number of particles involved into the process during the whole simulation, t_{CA} is the number of iterations per second measured during the simulation, $|M|$ is the cardinality of cellular array in the ACA. In [11] the physical time interval between two deposition acts t_{eff} is taken to correspond to the averaged time taken by adatoms to jump to the adjacent site.

$$\Delta t_{eff}^{-1} = \frac{k_B T}{2\pi\hbar} \exp(-\frac{E_c}{k_B T}).$$

The following scaling coefficients establish the correspondence between the ACA parameters and their physical counterparts.

1) *Length scale* $\mu_l = l\ \mathtt{m}$, l being the real linear size of the largest atom.

2) *Mass scale* $\mu_m^{(i)} = m_i\ \mathtt{kg}$, m_i being the real mass of an atom of ith species.

3) *Time scale* $\mu_t = \Delta t / \Delta t_{CA}$, Δt_{CA} being the computing time of an iteration measured during the simulation.

Moreover, CA-evolution being visualized allows the process to be observed in detail.

Example 3. *The simplified model of epitaxial growth of a silicon (Si) crystal.* The process [20] comprises two following actions: 1) adsorption of Si-atoms from an external flow; 2) diffusion of the absorbed atoms (adatoms) over the surface. If any atom deposited on the surface is adsorbed, then the rate coefficient $k_a = 1$. Being deposited on the surface layer by layer adatoms form pillars and islands of different height and size. The top atom on a pillar may diffuse to an adjacent site $(i + a, j + b)$, $a, b \in \{-1, 0, 1\}$, if $u(i + a, j + b) \leq u(i, j)$. Probability of such an action depends on bond strength between the adjacent adatoms, characterized by a constant $B = 0.05$ in the following way. If a site has n adjacent adatoms, then the probability of the diffusion is $p'_d = (4 - n) \cdot 0,05^n$. The choice among the sites whereto the adatom is allowed to move is equiprobable, which yields $p_d(k) = p'_d/(4 - n)$.

The process is simulated by an ACA$= \langle A, M, \Theta \rangle$ where $A = \{0, 1, \ldots\}$, $M = \{(i, j) : i, j = 0, \ldots, N\}$. A cell $(v, (i, j))$ corresponds to a site on a Si crystal surface, the thickness of the adsorbed layer being equal to v atoms. The transition rule $\Theta(i, j)$ is a superposition of Θ_{ads} responsible for absorbtion, and $\Theta_{diff}(i, j)))$ responsible for diffusion.

$$\begin{aligned} \Theta_{ads} &= \{(v_0, (i, j))\} \star \{(1, Gas)\} \overset{p_a}{\rightarrow} \{(v_0 + 1, (i, j))\}, \\ \Theta_{diff}^{(k)} &= \{(v_0, (i, j)), (v_k, \varphi_k(i, j))\} \overset{p_{d(k)}}{\rightarrow} \{(v_0 - 1, (i, j)), (v_k + 1, \varphi_k(i, j)\}, \\ & k = 1, 2, 3, 4. \end{aligned} \tag{25}$$

where $\varphi_k(i, j)$ is a kth neighbor of the cell (i, j).

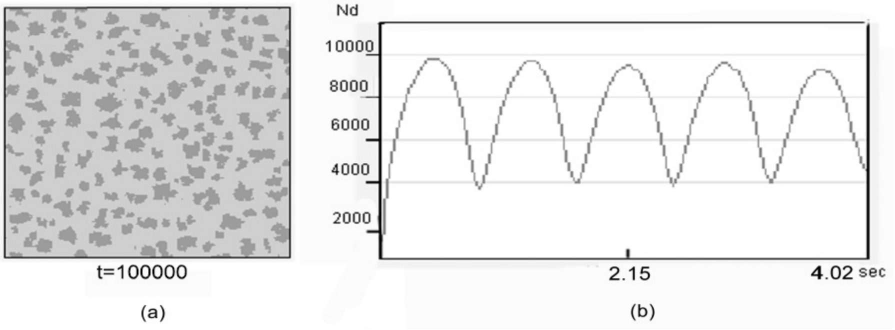


Fig. 4. Simulation results of epitaxial growth process. (a) Cellular array after t=100000 iterations (b), the dependence of islands perimeter $P(t)$ on real time. $|M| = 200 \times 200$. Intensity of gray color corresponds to the height of the island

A snapshot of the ACA evolution is shown in Fig. 4(a), where the formed islands on the crystal surface are seen. One of the important characteristic of the process is the dependence of total island perimeter $P(t)$ on time. The perimeter P is computed at each iteration as a number of pairs of adjacent cells having different states. During the process this parameter exhibits oscillations shown in Fig. 4(b), which are of interest for the researcher. The time scale is computed according to (24) based on the given rate of the deposition $ML = 1.4$ $\texttt{sec}^{-1}$ (monolayers per second $\mu_t = (ML \cdot |M|)^{-1} = 0.178 \cdot 10^{-4}$ $\texttt{s}$.

6 Conclusion

The problem of finding out the adequate correspondence between physical phenomenon and its CA-model is approached form the position of researcher who exploits CA simulation in studying certain natural phenomenon. Some general principles are formulated and, based on them, scaling coefficients are derived apart for three different types of CA-models. It is clear from the presented examples, that construction of a CA-model as well as finding out the correct interpretation of the simulation results, require a profound knowledge of physical fundamentals of the phenomenon under simulation. It affirms the fact that users of CA-simulations tools should be the physicists or engineers not experienced in CA theory. It makes the methodology of CA simulation yet more important and requested, thus implanting aspirations that the development of CA-models should include procedures of computing scaling coefficients.

References

[1] Toffolli, T. :(1984) Cellular Automata as an Alternative to (rather than Approximation of) Differential Equations in Modeling Physics. Physica D **10**. (1984) 117–127

[2] Wolfram, S.: A new kind of science. Wolfram Media Inc., Champaign, Ill., USA (2002)

[3] Bandman, O.: Comparative Study of Cellular Automata Diffusion Models. In: Malyshkin, V. (ed): Parallel Computer Technology (PaCT-1999). Lecture Nones in Computer Science, Vol.1662. Springer, Berlin. (1999) 395–404

[4] Malinetski, G. G., Stepantsov, M.E.: Modeling Diffusive Processes by Cellular Automata with Margolus Neighborhood. Zhurnal Vychislitelnoy Matematiki i Mathematicheskoy Phisiki. **36** (6) (1998) 1017–1021

[5] Weimar L. R., Boon J.-P.: Class of Cellular Automata for Reaction-Diffusion Systems. Phys. Rev., Vol.E, 49 (1994) 1749-1752.

[6] Boon J.P., Dab D., Kapral R., Lawniczak A. Lattice Gas Automata For Reactive Systems. Physics reports 273:22, 55-147, Elsevier Science (1996)

[7] Rothman, D. H., Zalesky, S.: Lattice-Gas Cellular Atomata. Simple Model of Complex Hydrodynamics. Cambridge University press, UK (1997)

[8] Succi, S.: The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Oxford University Press, NY (2001)

[9] Frish, U., d'Humieres, D., Hasslacher, B., Lallemand, P., Pomeau, Y., Rivet, J.P.: Lattice-Gas hydrodynamics in two and three dimensions. Complex Systems **1** (1987) 649–707

[10] Neizvestny, I. G., Shwartz, N.L., Yanovitskaya, Z. Sh., Zverev, A. V.: 3D-model of epitaxial growth on porous 111 and 100 Si Surface. Comp. Phys. Comm. **147** (2002) 272–275

[11] Gerish A., Lawniczak A.T., Budiman R.A., Ruda H.E., and Fuks H.: Lattice Cas Automaton Modeling of Surface Roughening in Homoepitaxial Growth in Nanowires. In: Proccedings of CCGEI 2003, Monreal, mai 2003 0-7803-7781-8/03, 001–004.

[12] Cornell S., Droz M. Chopard B.: Some Properties of the Diffusion-Limited Reaction $nA + mB \rightarrow C$ with Homogeneous and Inhomogeneous Initaial Conditions. Physica A 188, (1992) 332–336.

[13] Chopard B., Luthi P., Droz M.: Microscopic approach to the formation of Liesegang patterns. Journal of Statistical Physics. Vol. 76, N 1-2 (1996) 661-677

[14] Elokhin, V.I., Latkin E.I., Matveev A. V., Gorodetskii V.V.: Application of Statistical Lattice Models to the Analysis of Oscillatory and Autowave Processes in the Reaction of Carbon Monoxide Oxidation over Platinum and Palladium Surfaces. Kinetics and Catalysis **44** (5) (2003) 672–700

[15] Jansen, A.P.J.: An Introduction to Monte-Carlo Simulation of Surface Reactions. arXiv: cond-mat/0303028 v1 3 (2003)

[16] Boccara, N.: Modeling Complex Systems. Springer-Verlag, NY, Inc. (2004)

[17] Achasova, S., Bandman, O., Markova, V., Piskunov, S.: Parallel Substitution Algorithm. Theory and Application. World Scientific, Singapore (1994)

[18] Sahimi, M.: Flow phenomena in rocks: from continuum models to fractals, percolation, cellular automata and simulated annealing. Rev. in Modern Physics. **65** (4) 1393–1533(1993)

[19] Medvedev, Yu.G.: A Three-dimensional cellular automata model of viscous flow. Optoelectronics, Instrumentation and Data Processing **39** (3) (2003) 43–48

[20] Neizvestny1, I.G., Shvarts, N.L., and Yanovitskaya, Z. Sh.: Two-Dimensional Epitaxial Nucleation with Large Critical-Nucleus Size Russian Microelectronics. Science+Business Media LLC. **31** (2) 70–78 (2002)

[21] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Statistic Modeling. J.Chem.Phys. **21** 1087–1098 (1953)

Optimizing the creature's rule for all-to-all communication

Patrick Ediger and Rolf Hoffmann

Technische Universität Darmstadt
FB Informatik, FG Rechnerarchitektur
Hochschulstr. 10, 64289 Darmstadt, Germany
{ediger, hoffmann}@ra.informatik.tu-darmstadt.de

Abstract. A uniform cellular automaton is defined modeling creatures which can move around and which avoid collisions. The collision detection and resolution is performed in the current synchronous updating cycle using a neighborhood with the Manhattan distance of two. It is a subject for further discussion whether the proposed extension of a CA propagating intermediate results to their neighbors in order to reduce the overall computational effort is interesting from the theoretical and practical point of view.

The creatures' task is to exchange their information among each other (all-to-all-communication) in shortest time. The goal is to find automatically a very good creature's behavior for such a uniform multi-creature system. The creatures shall perform good on environments with borders as well as on environments without border (wrap-around). The behavior (algorithm) of a creature is defined by a state table which can be loaded into each creature. Best behaviors were evolved using a genetic procedure using a *training set* of initial configurations. Then they were ranked using a larger *ranking set* of initial configurations. The evolved creatures try to walk mainly on trails (a) which are orthogonal forming a sort of weaving pattern or (b) try to walk preferably at the borders or (c) in parallel to the borders. The found algorithms are very robust which was proved for another *robustness set* of initial configurations. The algorithms perform better for environments with borders meaning that they are using the borders to improve communication. In comparison random walkers perform much weaker than the evolved algorithms. In contrast to the evolved algorithms the communication of the random walkers is lowered by the borders.

1 Introduction

In former investigations we have tried to find out the best Cellular Automata (CA) rules for creatures in order to solve the creatures' exploration problem [1, 2]. The task was to visit all empty cells of an environment (CA grid with obstacles) in shortest time. We modeled the behavior by using a state automaton with two inputs and two outputs only in order to keep the complexity as low as possible. For automata with up to 7 states we were able to generate and evaluate by hardware support all relevant automata by a special enumeration technique which allows us to skip a priori over non relevant automata [3]. Relevant automata are a subset of all automata which can be coded by a state transition table. The subset is defined by certain conditions, e. g., equivalent automata under state and output coding permutation should only be enumerated once, or the state graphs should be strongly connected. This special enumeration technique allows us to enumerate and evaluate all algorithms until a certain complexity. Using hardware support we are able to accelerate the computation by about two orders of magnitude. This enumeration technique can help to yield optimal results for problems with low complexity or may be used as part of a heuristic.

Now we are addressing more complex multi-agent CA problems. The general goal of our research is to find out optimal (or sufficient good) local algorithms (defining the behavior of agents/creatures) in order to solve global tasks. At the moment we concentrate on problems with a low complexity in order to get fundamental insights about (1) how such agent algorithms can automatically be found in acceptable time (also using dedicated hardware architectures) and (2) what are the successful interaction principles between agents (techniques of cooperation, competition, communication) in order to fulfil the global task. As the search space for agent problems is very large we have to apply heuristic optimization techniques like simulated annealing, genetic algorithms or parallel meta-heuristics [9].

The presented problem is related to the creatures' exploration problem in the way how creatures can move. But the task is different: The creatures shall exchange their information (all-to-all) in shortest time. The goal is to find an optimal rule for the movements of the creatures in order to exchange their information as fast as possible. The information exchange is only possible when the creatures get close to each other and when they are forming certain defined local patterns (communication situations).

Examples of possible communication situations are shown in Fig. 1. In the cases a, b, c the creatures are directly in contact. But it is a matter of definition whether such situations allow communication. In a former investigation (to be published) we have defined the patterns d, e, f to be the only ones which allow communication through a mediator, which can receive incoming messages and transform them into outgoing messages. For this investigation we have defined the patterns a, b, c to be the only ones which allow communication: If a creature (shaded) detects another creature in front, it can read its information for updating its own state in accordance with the CA principle. Note that one creature may serve as a source of information for up to four creatures.

All-to-all communication is a very common task in distributed computing. The problem's specification can depend on many fixed or dynamic varying parameters like the number and location of nodes, the number and location of processes, the number, users and properties of the communication channels and so on. All-to-all communication in multi-creature systems is related to multi-

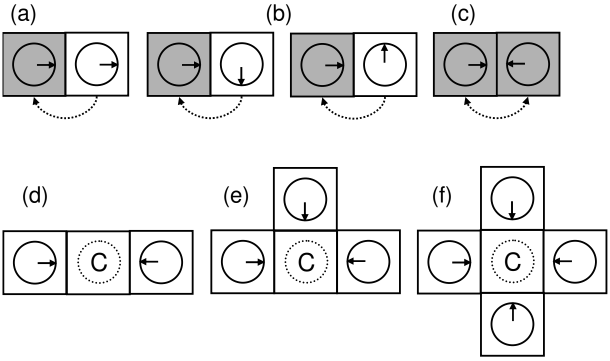


Fig. 1. Communication situations. Communication will only be allowed for the cases a, b, c, in which a creature reads information from another creature in front. Other situations like d, e, f using a mediator C are also reasonable, but not used here.

agent problems like finding a consensus [7], synchronizing oscillators, flocking theory or rendezvous in space [5], or in general to distributed algorithms with robots [4].

2 Modeling the multi-creature system

The whole system is modeled by a uniform 2D cellular automata. It models the environment with obstacles and borders and k uniform creatures which can move around. A creature can perform four different actions:

- R (turn Right): turn right only
- L (turn Left): turn left only
- Rm (turn Right and move): move forward and simultaneously turn right
- Lm (turn Left and move): move forward and simultaneously turn left.

The actions were defined in this way in order to keep the control automaton (Fig. 3, explained below) as simple as possible (only two input and two output values). A more powerful set of actions, including, e. g., move forward/backward with or without turn would require more output values.

A creature has a certain moving direction and it can only read the information from one cell ahead (the *front cell*). The creature investigates the front cell in order to evaluate the move condition m. If it detects a border cell or a creature or a conflict, the move condition is set to $m = 0$ (false) and otherwise to $m = 1$ (true). If the creature can move ($m = 1$) it performs Rm or Lm. If the creature cannot move ($m = 0$) it will perform R or L.

The cell state is defined as follows:

$$state = (type, d, s) \qquad //d = direction, s = control\ state$$

$$type \in \{EMPTY, OBSTACLE, CREATURE\}$$

$$d \in \{toNorth, toEast, toSouth, toWest\} = \{0, 1, 2, 3\}$$

$$s \in \{0, 1, 2, \ldots n-1\}$$

The neighborhood is defined by the Manhattan distance of two (i. e., MOORE neighborhood plus the four cells at straight distance two from the center: NN = NorthNorth, EE = EastEast, SS = SouthSouth, WW = WestWest). In the case that a cell is of type EMPTY the used neighborhood is only the von Neumann neighborhood.

The moving of a creature was defined and implemented in the CA model using the following rules (described in a simplified pseudo code; only the changing values are stated). *RuleS* is the function which defines the next control state. *RuleY* is the function which determines the activator y (Fig. 3) which in turn determines the change of the direction. Thus the function *RuleY* in combination with the possibility to increment or decrement d defines the whole transition rule for d.

```
if (type==EMPTY) then
  sum = (N.type==CREATURE) & (N.d==toSouth) +
   (E.type==CREATURE) & (E.d==toWest) +
   (S.type==CREATURE) & (S.d==toNorth) +
   (W.type==CREATURE) & (W.d==toEast))
  if (sum==1) then m=1 else m=0; //move condition
  if (m==1) then
    type ← CREATURE
    if (N.d==toSouth) then
      d ← $(d + (-1)^{RuleY(m=1,N.s)})$ mod 4
      s ← RuleS(m=1, N.s)
    endif
    if (E.d==toWest) then
      d ← $(d + (-1)^{RuleY(m=1,E.s)})$ mod 4
      s ← RuleS(m=1, E.s)
    endif
    if (S.d==toNorth) then
      d ← $(d + (-1)^{RuleY(m=1,S.s)})$ mod 4
      s ← RuleS(m=1, S.s)
    endif
    if (W.d==toEast) then
      d ← $(d + (-1)^{RuleY(m=1,W.s)})$ mod 4
      s ← RuleS(m=1, W.s)
    endif
  endif m
endif EMPTY

if (type==CREATURE) then
  m = NOT(
   ((d==toNorth) & ((NN.type==CREATURE & NN.d==toSouth) |
   (NW.type==CREATURE & NW.d==toEast) |
   (NE.type==CREATURE & NE.d==toWest)) |
   ((d==toWest) & ((WW.type==CREATURE & WW.d==toEast) |
   (NW.type==CREATURE & NW.d==toSouth) |
   (SW.type==CREATURE & SW.d==toNorth)) |
   ((d==toSouth) & ((SS.type==CREATURE & SS.d==toNorth) |
   (SW.type==CREATURE & SW.d==toEast) |
   (SE.type==CREATURE & SE.d==toWest)) |
   ((d==toEast) & ((EE.type==CREATURE & EE.d==toWest) |
   (NE.type==CREATURE & NE.d==toSouth) |
   (SE.type==CREATURE & SE.d==toNorth))
   )
  if (m==1) then type ← EMPTY endif
  else
    d ← $(d + (-1)^{RuleY(m=0,s)})$ mod 4
    s ← RuleS(m=0, s)
  endif m
endif CREATURE
```

A conflict occurs when two or more creatures want to move to the same front cell. Conflicts are resolved in one generation (during the current clock cycle) [10].Therefore the neighborhood with distance two is necessary. If the neighborhood is restricted to von Neumann, than two generations were necessary to resolve the conflict.

In a software and hardware implementation the above rules can be simplified. The idea behind is that the move signal which is computed as an intermediate value in an EMPTY front cell can also be used in the 4 adjacent CREATURE cells. Because the move signal can at the same time trigger the CREATURE to be deleted and the EMPTY front cell to become a CREATURE. Generally speaking the moving of information under conservation (e. g., like the moving of tokens in a Petri net) requires also in a CA the simultaneous deletion of the information particle on one site and its generation on another site.

In our hardware implementation we followed this idea: Each creature which wants to visit the same front cell sends a request signal and awaits a grant signal which is sent back to the creature to be selected. No grant signal is sent back if more than one creature requests. Each cell contains logic (from the hardware view) or an additional function (from the mathematical view) which generates feedback signals for arbitration (Fig. 2). By this technique the neighborhood of a creature (the front cell) is indirectly extended to all neighbors of the front cell. Therefore a creature can be informed through the front cell that another creature is either two cells ahead or diagonal in front. The arbitration signal is only influencing the next state of the creature but is not further propagated thereby avoiding very long propagation chains or possible asynchronous oscillations.

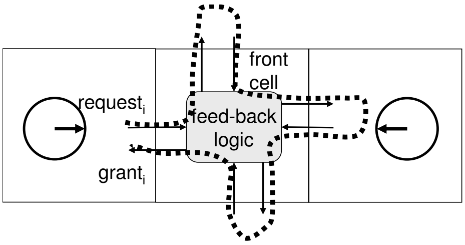


Fig. 2. Asynchronous feed-back logic used for arbitration.

In order to model the distribution of information we are using an additional bit vector in the cell state with k bits which is stored in each creature. At the beginning the bits are set mutually exclusive (bit(i)=1 for creature(i)). When a creature A detects another creature B in front of it, it will OR the bit vector of B into its own A bit vector. The task is successfully solved when the bit vectors of all creatures obtain $11\ldots1$.

A creature is implemented by a *control machine* (MEALY automaton) and an *action machine* (MOORE automaton) which is controlled by the control machine (Fig. 3). The state of the control machine is the control state s. The state of the action machine is the direction r. The action machine reacts on the control signal (*activator*) d. If $d = 1$ the creature turns to the right ($r := r + 1$), otherwise to the left ($r := r - 1$). The behavior of the action machine is predefined and fixed.

The behavior defined by the control machine (also called algorithm for short in this context) is changeable and can be configured by loading a state transition table. The number of different control states is n. Input to the state table is s and the move signal m. Output of the state table is the control signal $d = RuleY$ and the next state $s' = RuleS$. Note that the union of the control machine with the action machine results in a MOORE (of subtype MEDWEDJEW)) automaton. Note that the MEDWEDJEW automaton is the common automaton used in the standard CA model.

To summarize we are describing the moving of creatures by appropriate rules according to the standard CA model with an extended neighborhood. The neighborhood can virtually be reduced if the move signal is computed by the front cell and also used by the surrounding cells. In general this idea leads to an interesting extension of the standard CA model. If such an extension should be allowed is a good point for discussion.

A state transition table with current input x, current state s, next state s' and current output y is shown in Fig. 4. It can be represented by concatenating the contents to a string, e. g.:

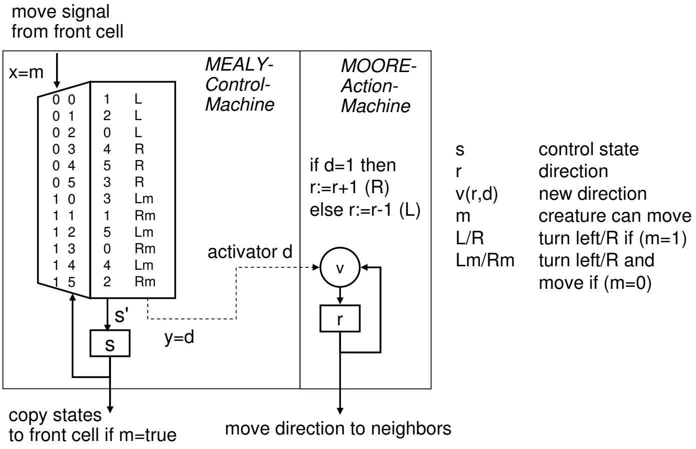


Fig. 3. The main parts of a cell: Table driven control machine controlling action machine. The next state is defined by RuleS (the left column) and by RuleY (the right column)

1R5L3L4R5R3L-1Lm2Rm3Rm4Lm5Rm0Lm
1R5L3L4R5R3L-1L2R3R4L5R0L (simplified form)

(a)

x	0						1					
s	0	1	2	3	4	5	0	1	2	3	4	5
s',y	2,R	5,L	3,L	5,L	2,R	5,R	3,Lm	4,Lm	0,Lm	1,Rm	5,Rm	2,Rm
i	0	1	2	3	4	5	6	7	8	9	10	11
j	0,1	2,3	4,5	6,7	8,9	10,11	12,13	14,15	16,17	18,19	20,21	22,23

(b)

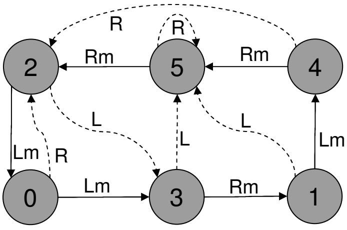


Fig. 4. (a) A state table, defining the behavior (algorithm) of a creature and (b) the corresponding state graph. *Input* $= x = m$ (move condition). *Output* $= y = d$ (activator). Solid arcs are used for $m = 1$, dashed arcs are used for $m = 0$.

The state table can be represented more clearly as a state graph (Fig. 4b). If the state machine uses n states, we call such an algorithm n-state algorithm. The number of M of all algorithms (MEALY type) which can be coded by a table oriented state machine is

$$M = (\#s \times \#y)^{(\#s \times \#x)}$$

where $n = \#s$ is the number of states, $\#x$ is the number of different inputs and $\#y$ is the number of different outputs. Note that M increases dramatically, especially with $\#s$, which makes it very difficult or even impossible evaluate the quality of all algorithms in reasonable time (e. g. for $\#s > 6$ with $\#x = \#y = 2$).

3 Investigations and results

Random walk. In order to compare the quality of the algorithms to be found we used random walkers with the same actions. Two environments were distinguished: (1) grid with border and (2) grid without border (wrap-around). 300 random walks on 50 random initial configurations were performed for each of the two environments. In the first environment 1,882 generations on average were needed to solve all of the test cases. In the second environment (wrap-around) 2,199 generations were needed. It is interesting to note that the random walkers can communicate better if they are not disturbed by the borders. We will show later, that in the case of using optimized behaviors, just the opposite will be the case. Then the creatures make use of the borders to improve their communication. The following results will also show that the random walkers are significantly slower than the creatures with optimized behaviors.

Simplification of the problem. To solve the problem very general either theoretical or practical with respect to all interesting parameters is very difficult. Therefore we have simplified our investigation by using heuristics and some constants instead of parameters. The grid size was set to 33 by 33 and the number of creatures was set to $k = 16$. From former investigations of multi-creature systems we know that a number of creatures between approx. 8 and 64 can lead to good synergy effects and a sufficient number of conflicts which are required here. The ultimate goal is to find the optimal behavior on average (robust algorithm) for all possible initial configurations. An initial configuration is defined by the start position and direction of the creatures and by the environment (with border or wrap-around). Thus the searched behavior shall be able to cope with both types of environments.

As we cannot test all possible initial configurations we will be satisfied if we can find best behaviors for some test sets comprising a "sufficient" amount of initial configurations. We defined several test sets:

1. **(Training Set)** 10 randomly generated initial configurations which are used in the genetic procedure (see below). 5 of them use a grid with border, 5 of them use a grid with wrap-around.
2. **(Ranking Set)** 100 randomly generated initial configurations which are used to check and rank the quality of the algorithms found by the genetic procedure. 50 of them use a grid with border, 50 of them use a grid with wrap-around.

3. **(Robustness Set)** 80 initial configurations (40 without border, 40 with border) with the same amount of cells but with grid sizes of 1089 × 1, 363 × 3, 121 × 9 and 99 × 11, and with 16 randomly distributed creatures were defined in order to test the algorithms for robustness.

Genetic Procedure. As the search space for different behaviors is very large we are not able to check all possible behaviors. Therefore we used a genetic procedure and tried to find the best behavior within a reasonable computational time limit.

The fitness of a uniform multi-creature system (using a specific common behavior) is defined as the number of generations which are necessary to distribute (all-to-all) the information, averaged over all initial configurations under test. In other words we search for state algorithms which can solve the problem with a minimum number of generations.

The behavior is described by a state table (Fig. 4) using 6 control states, 2 inputs ($x = 0/1$) and 2 outputs ($y = 0/1$). The string representation of the state table defines the genome (individual, possible solution). P Populations of N individuals are updated in each iteration (optimization cycle). During each cycle M off-springs are produced in each population. The union of the current N individuals and the M off-springs are sorted according to their fitness and the N best are selected to form the next population. An offspring is produced as follows:

1. (Get Parents) Two parents are chosen for each population. Each parent is chosen from the own population with a probability of p_1 and from an arbitrary other population with the probability of $(1 - p_1)$.
2. (Crossover) Each new component (s'_i, y_i) of the genome string is taken from either the first parent or the second parent with a probability of 50%. This means that the tuple (next state, output) for the position i=(input, state) is inherited from any parent.
3. (Mutation) The string being modified by the crossover is afterwords mutated with a probability of p_2. If a mutation shall be performed, an arbitrary position j is chosen and a new value (randomly chosen from the set of valid values) is replacing the existing one. Thereby either the next state or the output is changed at position i.

The algorithms were optimized using $P = 7$ populations with $N = 100$ individuals each, $M = 10$ off-springs, $p_1 = 2\%$ and $p_2 = 9\%$. 6000 iterations were performed starting with randomly generated state algorithms. The fitness was tested for each offspring by simulating the multi-creature systems with the training set.

Evaluation. The 700 best behaviors which had evolved after 6000 iterations were used for the additional ranking test set. Thereby the best 100 algorithms were selected ordered by (1) their success rate (number or percentage of initial configurations which were successfully processed) and (2) their speed (number of generations). The top 5 are listed in Table 1 upper part. All of them were completely successful (100 out of 100 configurations) and the best algorithm needed 773 generations on average (mean of 50 configurations with border and 50 without).

	Algorithm		Ranking Set			Robustness Set		
			Success Rate (max. 100)	Mean Generations	Communication Spots (a+b+c)	Success Rate (max. 80)	Mean Generations	Communication Spots (a+b+c)
Best of Ranking Set	A_1	2R5L3L5L2R5R-3L4L0L1R5R2R	***100***	***773.21***	209.9	80	1905.61	378.26
	B_1	1R3L5L4L2R1R-3L4L0L1R5R2R	***100***	***775.58***	215.82	60	1329.35	327.53
	C_1	5L5L5L3R2R4R-4R2R3L5L1L0R	***100***	***775.65***	116.76	60	1601.12	183.78
	D_1	5L3R4L0L3R3R-2R0L4L1L5R3R	***100***	***778.31***	111.13	60	1382.75	136.82
	E_1	4L3R5L4L1R5L-2R0L4L1L5R3R	***100***	***782.52***	197.82	68	2477.79	331.06
Best of Robustness Set	A_2	2R5L3L5L2R5R-3L4L0L1R5R2R	100	773.21	209.9	***80***	***1905.61***	378.26
	B_2	4L3R3R4L5R5L-2R0L4L1L5R3R	99	788.9	251.36	***79***	***1607.33***	409.85
	C_2	0L3L0R2L2L1R-5L2L4R1R0R3L	100	783.88	287.52	***79***	***1657.99***	443.86
	D_2	1R3L2L0R3L5R-5L2L4R1R0R3L	100	796.32	219.21	***79***	***1968.6***	385.7
	E_2	4R1R0L3R3L0L-5R2L4R0L3L1R	99	799.23	235.47	***78***	***1476.56***	361.06

Table 1. The top 5 algorithms of the Ranking Set and of the Robustness Set ordered by (1) "Success Rate" (number of successfully solved configurations) and (2) "Mean Generations" (number of generations averaged over all successful configurations). The column "Communication Spots" represents the mean number of cells from which information was read.

Figure 5 shows the distribution of generations with respect to the 100 initial configurations for the best algorithm A_1. Configurations with a border can be solved faster with a lower variance than the configurations with wrap-around. In the 33 × 33 environment with borders the algorithm A_1 needed 474 generations (averaged over the 50 configurations) whereas the random walkers needed 2,199 generations on average. In the environment with wrap-around the algorithm A_1 needed 1,072 generations (averaged over the 50 configurations) whereas the random walkers needed 1,882 generations on average.

The 700 best algorithms obtained by the genetic procedure were also used for an additional robustness test using the robustness test set. The ordering scheme is the same as the one used for the ranking set. The top 5 are listed in Table 1 lower part. Only one of them $A_2 = A_1$ was completely successful (80 out of 80 configurations) and the best algorithm A_2 needed 1,906 generations on average (mean of 40 configurations with border and 40 without). The following three algorithms B_2, C_2, D_2 cannot solve one configuration: It is one of the configurations of size 1089 × 1. Algorithm E_2 cannot solve two configurations of size 1089 × 1.

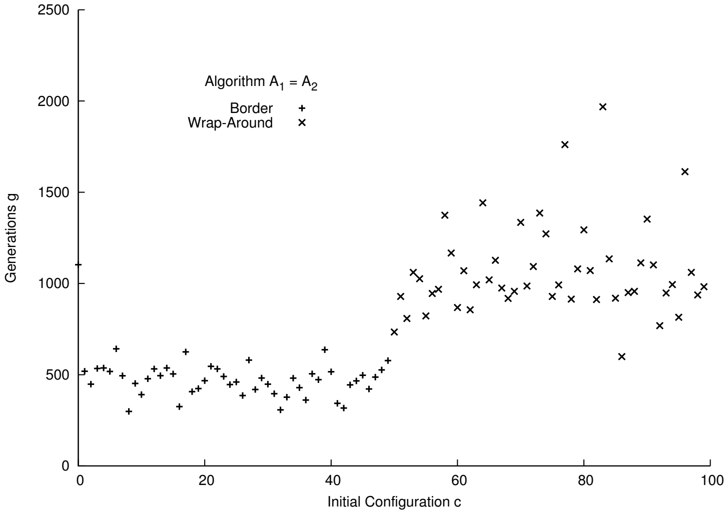
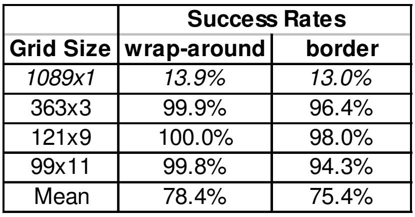


Fig. 5. The number of generations needed by algorithm $A_1 = A_2$ for each of the 100 initial configurations of the Ranking Set. The initial configurations $0 \dots 49$ are with border, the configurations $50 \dots 99$ are with wrap-around. The creatures solve the problem faster (474 on average) with border than with wrap-around (1072 on average).

Further investigations have shown (Table 2) that environments of size $n \times 1$ are in general more difficult to solve with the evolved algorithms. The success rates for "wrap-around" are slightly better than for "border".



	Success Rates	
Grid Size	**wrap-around**	**border**
1089x1	*13.9%*	*13.0%*
363x3	99.9%	96.4%
121x9	100.0%	98.0%
99x11	99.8%	94.3%
Mean	78.4%	75.4%

Table 2. Overall success rates of the Robustness Set, separate for each grid size.

Figure 6 shows how many of the best 100 algorithms ordered by the robustness test can solve a certain number of the 80 initial configurations of the robustness set. It can be seen that less than 50-60 configurations are easy to solve whereas more configurations are difficult to solve. The analysis revealed that the difficult environments were in particular the environments 1089 × 1.

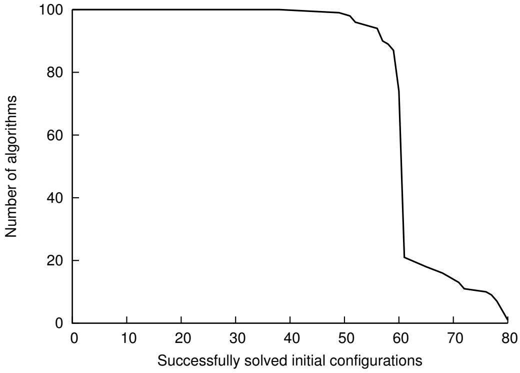


Fig. 6. Curve showing how many of the 100 algorithms can solve a certain number of initial configurations in the Robustness Set.

Global Behavior. All algorithms which could successfully communicate in all the 100 configurations show a similar global behavior: In the environments with wrap-around the creatures form trails on which they preferably move. These trails are forming a sort of a grid or weaving pattern. This grid is slightly turned to the right or to the left (Fig. 7). The communications take place on the trails and on the crossings of the trails. Similar patterns have emerged for all the algorithms out of the top 100 we have checked. Analyzing the behavior of a single creature showed that it is following a trail which forms a spiral with wrap-around in the torus. Thus several creatures are forming the weaving pattern. When creatures meet they are slightly deviated or they turn by 90 degrees. But the algorithms are robust enough holding the creatures more or less on the trails.

The trail patterns which are emerging in the environments with borders look similar but in addition trails and many communication spots along the borders are establishing. A communication spot (marked white in Fig. 7, 8) is a cell which at least once is used as an information source. In some of the multi-creature systems also trails which run in parallel to the borders (Fig. 7b, 8b) can be observed. In Fig. 8 appear also little squares in which creatures are circulating around. These squares may be interpreted as centers of information which stay at their place until they are kicked into regular trails.

4 Conclusion

We have explained in detail CA rules which allow to model moving creatures with conflict resolution. The moving behavior of a creature is defined by a state table. As an example we have modeled all-to-all communication between creatures. The best behaviors were evolved by a genetic procedure on a Learning Set of initial configurations comprising 50% environments with border and 50% with wrap-around. A larger Ranking Set and a larger Robustness Set of initial configurations

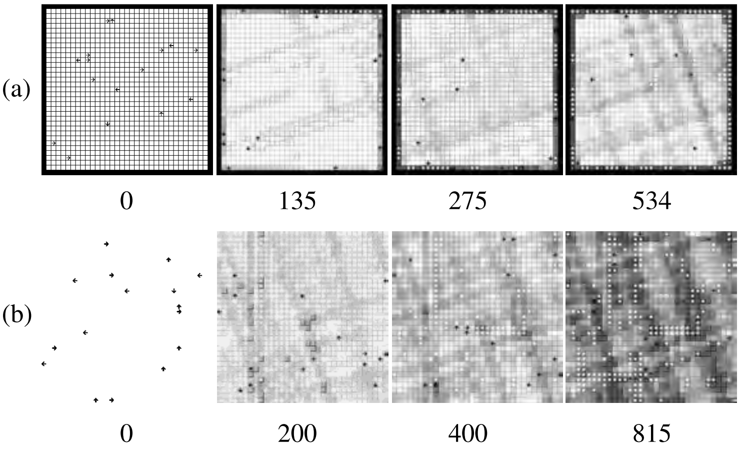


Fig. 7. Patterns emerged by Algorithm $A_1 = A_2$ (see also Tab. 1). The numbers indicate the generations of the CA when the situations were observed. The rightmost images show the final generation when the algorithms accomplished all-to-all communication.

were used to evaluate and order the evolved algorithms. The best algorithm is very robust and performs very well on all the tested environments.

The algorithms perform better on environments with border compared to environments with wrap-around because the movements along the borders improve the communication. Compared to random walkers the evolved algorithms perform much better. In contrast to the random walkers the evolved algorithms perform better on environments with borders whereas random walkers perform better on the other environments with wrap-around, meaning that random walkers are not able to take advantage out of the borders.

The evolved algorithms emerge mesh like trailing patterns which are reminding us of weaving structures. In addition trails which are in parallel to the borders have emerged.

Further investigations are considered with different actions, a different number of control states, time-shuffled algorithms [8], non-uniform creatures [6], specialized fixed communication cells, and complete quality tests or formal proves for small environments.

References

[1] Mathias Halbach, Rolf Hoffmann: Optimal behavior of a moving creature in the cellular automata model. 8th International Conference Parallel Computing Technologies, PaCT 2005, 129–140

[2] Mathias Halbach, Wolfgang Heenes, Rolf Hoffmann, Jan Tisje: Optimizing the behavior of a moving creature in software and in hardware. 6th International Conference on Cellular Automata for Research and Industry, ACRI 2004, 841–850

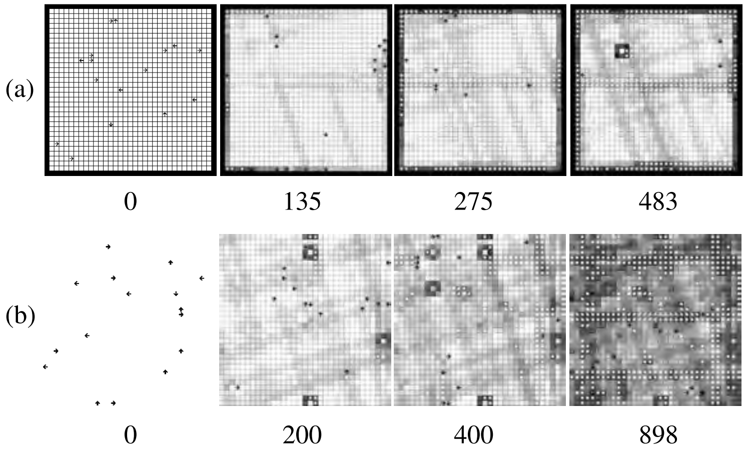


Fig. 8. Patterns emerged by Algorithm B_2 (see also Tab. 1) on the same initial configurations as in Fig. 7.

[3] Mathias Halbach: Algorithmen und Hardwarearchitekturen zur optimierten Aufzählung von Automaten und deren Einsatz bei der Simulation künstlicher Kreaturen. To be published in May 2008, Technische Universität Darmstadt

[4] Giuseppe Principe, Nicola Santoro: Distributed algorithms for autonomous mobile robots. 4th IFIP Int. Conf. on TCS, 2006, 47–62

[5] J. Lin, A. S. Morse, B. D. O. Anderson: The multi-agent rendezvous problem. An extended summary. Cooperative Control, Vol. 309, 2005, 257–289

[6] Patrick Ediger, Rolf Hoffmann, Mathias Halbach: Is a non-uniform system of creatures more efficient than a uniform one? Workshop on Nature Inspired and Distributed Computing at IPDPS 2008

[7] Reza Olfati-Saber, J. Alex Fax, Richard M. Murray: Consensus and cooperation in networked multi-agent systems. Proceedings of the IEEE, Vol. 95, 2007, 215–233

[8] Patrick Ediger, Rolf Hoffmann, Mathias Halbach: How efficient are creatures with time-shuffled behaviors? 9th Workshop on Parallel Systems and Algorithms (PASA), 2008, 93–103

[9] Enrique Alba: Parallel Metaheuristics: A New Class of Algorithms. John Wiley & Sons, NJ, USA, 2005

[10] Mathias Halbach, Rolf Hoffmann: Solving the exploration's problem with several creatures more efficiently. 11th International Conference on Computer Aided Systems Theory (EUROCAST), 2007, 596–603

.

A Java based three-dimensional cellular automata simulator and its application to three-dimensional Larger than Life

Mitsuru Matsushima[1], Katsunobu Imai[2], Chuzo Iwamoto[2], and Kenichi Morita[2]

[1] NEC Corporation
[2] Graduate School of Engineering, Hiroshima University
imai@iec.hiroshima-u.ac.jp

1 A three-dimensional cellular automata simulator

There are many simulation programs of various kinds of cellular automata (CA). In the two-dimensional case, there are some "killer applications" such as Golly [9]. Such a multi-purpose simulator enables constructing rules and configurations without programming any new simulation tool.

In the three-dimensional case, there are also many simulation tools [2, 5] and even Mathematica also serves one as an example. But it is difficult to study three-dimensional cellular automata employing existing simulation tools. Because the configurations of three-dimensional CAs apt to be large and designing a user interface of intuitive and comprehensive manipulation of them is quite difficult. So quite a few tools are intend to simulate a specific type of three-dimensional CA.

Once some of the authors had planned to build a three-dimensional cellular automaton with a complicated rule and configurations [6], a special simulation tool was programmed at first. We used Common Lisp and QuickDraw 3D [11] for rapid prototyping. Although the simulator was easy to modify and we divert it to built three-dimensional array grammars [7], it is not so easy to maintain by other research groups. Because the programming environment is not popular and works only on the MacOS.

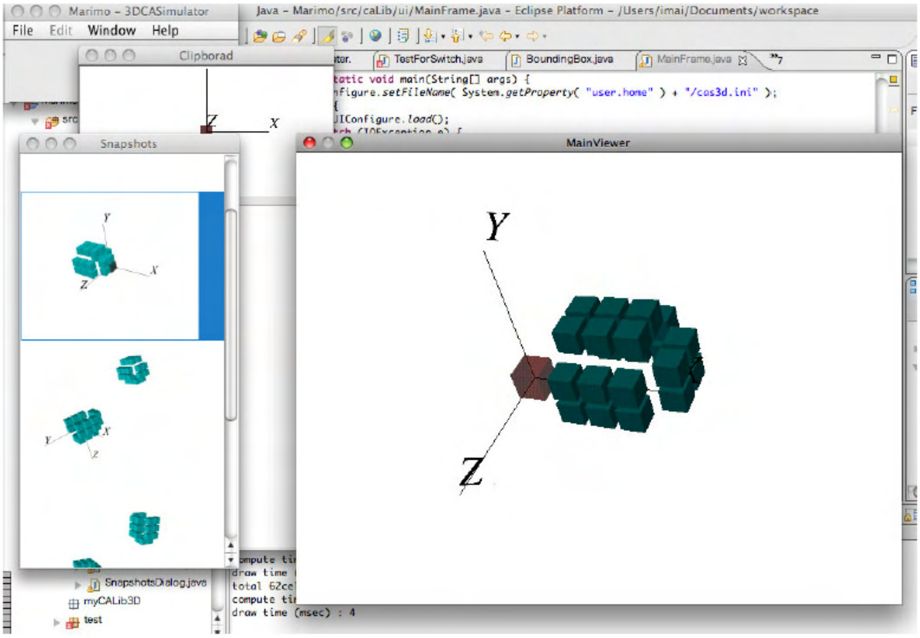
So we intend to build a Java based multi-purpose three-dimensional cellular automata simulator. We used the Java3D [10] and Eclipse [8].

Figure 1 is a screen snapshot of the current prototype of the simulator. The shape of neighborhood is variable and some user interfaces such as clip boards are equipped so far.

2 Three-dimensional Larger than Life cellular automata

To show the potential of our simulator, we tried to find some interesting rules and configurations of three-dimensional Larger than Life CAs.

Larger than Life $L = (r, [\beta_1, \beta_2], [\delta_1, \delta_2])$ is a natural generalization of the game of life. Where r is the radius of neighborhood, $[\beta_1, \beta_2]$ is the range of the



```
tatic void main(String[] args) {
nfigure.setFileName( System.getProperty( "user.home" ) + "/cas3d.ini" );
{
UIConfigure.load();
tch (IOException e) {
```

```
ompute ti
draw time
total 62ce
compute ti
draw time (msec) : 4
```

Fig. 1. A screen snapshot of the simulator.

total number of live neighborhood cells for giving birth to the center cell ($0 \longrightarrow 1$), $[\delta_1, \delta_2]$ is the range for survival of the center cell ($1 \longrightarrow 1$) [3]. The game of life can be denoted by $(1, [3, 3], [2, 3])$. Impressive patters such as Bugs and Bosco are found on the Larger than Life of large neighborhood (radius 5) [3] and it also has high computing efficiencies [4].

As far as the game of life, there are three-dimensional extensions. In contrast to the two-dimensional case, there are many combinations of the range of birth and survival. Bays defined a condition for three-dimensional game of life and find some candidates [1].

We tried to find some life like rules and patterns in the case of $r = 2$ and found several rule candidates and interesting patterns including gliders. Fig. 2 shows a glider patters on the rule $(2, [14, 15], [10, 15])$. We are also trying to find some rule candidates and functional patterns for the case of larger radius.

References

[1] Bays, C., Candidates for the Game of Life in Three Dimensions, Complex Systems, 1, 373-400, (1987).

[2] Bays, C.: The game of three dimensional life, `http://www.cse.sc.edu/~bays/d4d4d4/` .

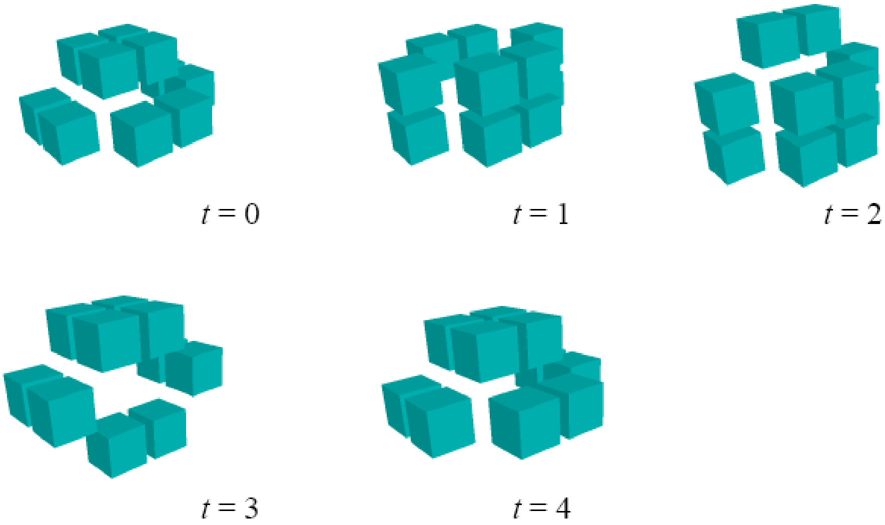


Fig. 2. A glider of the rule $(2, [14, 15], [10, 15])$.

[3] Evans, K.M.: Larger than Life: Digital Creatures in a Family of Two-Dimensional Cellular Automata, Discrete Mathematics and Theoretical Computer Science Proceedings AA (DM-CCG) 177–192 (2001).

[4] Evans, K.M.: Is Bosco's Rule Universal?, Proceedings of Machines, Computations, and Universality, Saint Petersburg, Lecture Notes in Computer Science 3354, Springer, 188–199 (2004).

[5] Freiwald, U. and Weimar, J.R.: JCASim–a Java system for simulating cellular automata, Theoretical and Practical Issues on Cellular Automata (ACRI 2000), S. Bandini and T. Worsch (eds), Springer Verlag, London, 47–54 (2001).

[6] Imai, K., Hori, T. and Morita, K.: Self-reproduction in three-dimensional reversible cellular space, Artificial Life 8 155–174 (2002).

[7] Imai, K., Matsuda, Y., Imamoto, C., and Morita, K.: A three-dimensional uniquely parsable array grammar that generates and parses cubes, Electric Notes in Theoretical Comupter Science 46 (2001) 16 pages, (Proceedings of the 8th International Workshop on Combinatrial Image Analysis (IWCIA), Philadelphia (2001) 351–366).

[8] Eclipse - an open development platform, `http://www.eclipse.org/` .

[9] Golly Game of Life, `http://golly.sourceforge.net/` .

[10] Java3D, `http://java.sun.com/javase/technologies/desktop/java3d/` .

[11] QuickDraw 3D, `http://en.wikipedia.org/wiki/QuickDraw_3D` .

.

Solving cellular automata problems with SAGE/Python

Iztok Jeras

Faculty of Computer Science, University of Ljubljana
iztok.jeras@rattus.info,
http://rattus.info/al/al.html

Abstract. This article proposes the use of high level programing languages for quick experimentation, verification of algorithms, data visualization and as a replacement for pseudo code in scientific articles. Example SAGE/Python code for a few CA related problems is provided.

1 Introduction

This article evaluates SAGE `http://www.sagemath.org` a relatively new mathematical software. SAGE is an open source package based on Python that integrates different existing software and ads its own code. It was chosen over other tools, because it provides:

- good tools for integer mathematics (compared to Matlab)
- an arguably large set of libraries (compared to raw C)
- easy creation of interfaces (OS, other software, graphics)

I started the evaluation with a set of criteria to be observed. In approximately two weeks I wrote a short software package and evaluated some of the proposed criteria. The focus was on theoretical CA problems instead of practical applications, where programs are already common (Life simulators, ...). The article lists the package source code and a set of examples showing its features.

Note, that the source code included in the article was created in a short time frame as a proof of concept and is not ready for production purposes.

2 Evaluation criteria

To be able to evaluate the usefulness of SAGE/Python, I made a list of problems to be solved and requirements to be met. Different criteria must be met for known practical and new theoretical problems.

CA simulators are a common example of practical problems, where large amounts of data are processed and visualized. Common requirements are: efficient resource consumption (memory, processing power, processing time), scalability and portability (over present and future platforms).

Software is used in theoretical CA research to quickly create simulators of formally defined systems and implementations of algorithms. Such software can than be used to pass knowledge to other researches or students. Common requirements are: availability of libraries, easy learning curve, compact and readable code.

The main criteria this article tries to evaluate is the appropriateness of SAGE as an alternative for pseudo code.

3 Description of the included source code

The source code included in the appendix defines a set of classes and functions for 1D CA with the purpose of solving the next problems:

- definition of 1D CA rule and lattice
- global state transitions forward and backward in time
- checking for garden of eden configurations

The computation of preimages was discussed by Wuensche [1] and Mora, Juárez and McIntosh [2], the implemented algorithm was described by Jeras and Dobnikar [5]. McIntosh [3, 4] also describes de Bruijn and subset diagrams.

3.1 1D CA definition

The definition of a cellular automaton is divided into two classes (rule and lattice), that can be arbitrary combined.

The rule class `CA1D_rule` is used to store the initialization parameters (`k` - cell states, `m` - neighborhood size, `r` - rule number) and computed parameters (`f` - local transition function table, `D` - de Bruijn diagram matrices, `Sf` - forward subset diagram table).

```
sage: ca = CA1D_rule (2, 3, 110); ca
Cellular Automaton:
  states    = 2
  neighbors = 3
  rule      = 110
```

The lattice class `CA1D_lattice` describes the size of the CA lattice and boundary conditions. In addition to parameters (`ca` - rule object, `C` - configuration, `sh` - output cell position in its neighborhood, `b` - boundary type) there are methods (`next()` - compute evolution step, `prev()` - compute preimages, `isGoE()` - check for Garden of Eden configurations) used to perform computations on the CA.

```
sage: lt = CA1D_lattice (ca, [0,0,0,1,0,0,1,1,0,1,1,1,1,1], sh=1)
sage: lt
[0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1]
```

3.2 Local transition function and global transitions

When a CA1D object is created the rule number is transformed into a table describing the local transition function. Element `0` in the list is the output of the local function with the neighborhood `000` as input.

```
sage: ca.f
[0, 1, 1, 1, 0, 1, 1, 0]
```

The next code prints the lattice state for 5 time steps. Since the example configuration is the ether for rule 110, common triangular patterns can be seen. A `'cyclic'` and an `'open'` lattice are processed.

```
sage: for i in range(5) : lt.next(); print lt
[0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1]
[0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1]
[1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1]
[0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0]
[1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0]

sage: lt.b = 'open'
sage: for i in range(5) : lt.next(); print lt
[1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0]
   [0, 1, 0, 0, 1, 1, 0, 1, 1, 1]
      [1, 0, 1, 1, 1, 1, 1, 0]
         [1, 1, 0, 0, 0, 1]
            [1, 0, 0, 1]
```

3.3 de Bruijn diagram and preimages

The de Bruijn diagram is represented as a list of matrices, one for each cell state.

```
sage: ca.D[0]                 sage: ca.D[1]
[1 0 0 0]                     [0 1 0 0]
[0 0 0 0]                     [0 0 1 1]
[1 0 0 0]                     [0 1 0 0]
[0 0 0 1]                     [0 0 1 0]
```

This matrices are used to compute preimages. The `prev()` method returns a list of preimages. In the first example for `'cyclic'` and than for `'open'` (unrestricted) boundaries.

```
sage: lt = CA1D_lattice(ca,[0,0,0,1,0,0,1,1,0,1,1,1,1,1],sh=1)
sage: lt.prev()
[[1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1],
 [1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1]]

sage: lt.b = 'open'; lt.prev()
[[1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1],
 [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0],
 [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0],
 [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1],
 [1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0],
 [1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1],
 [1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1]]
```

For non cyclic lattices the preimage configuration is larger than the input configuration.

3.4 Subset diagram and Garden of Eden configurations

The subset diagram is build with the creation of the rule object. Each line represents the subset transitions for one of the cell states.

```
sage: ca.Sf
[[0, 1, 0, 1, 1, 1, 1, 1, 8, 9, 8, 9, 9, 9, 9, 9],
 [0, 2, 12, 14, 2, 2, 14, 14, 4, 6, 12, 14, 6, 6, 14, 14]]
```

The included code supports checking for GoE configurations only for `'open'` boundary conditions. The subset diagram is used as a finite state automaton where the initial state is the full set and the accepting state (for the GoE regular language) is the empty set.

```
sage: lt = CA1D_lattice (ca, [0,1,0,1,0], sh=1); lt.isGoE()
True
```

4 Overview and conclusion

After writing the listed code, I can say that SAGE and Python are appropriate tools for quick implementations. I was able to write the code without much prior Python knowledge, and the implementation of algorithms is short and compact enough to be included in an article.

The listed code can be improved in many ways. By adding complexity (libraries) and optimizations (written in C) larger problems could be processed. Packages like NetworkX[1] and Python Imaging Library[2] can be used to draw basins of attraction and create PNG images of CA evolution. Matrix multiplication can be optimized using the NumPy package[3]. The SAGE project by itself ads a lot of useful tools for theoretical analysis of CA, I was mostly interested in its combinatorics capabilities.

The appropriateness of SAGE and Python as an alternative to pseudo code is not obvious. The main problem is the fast evolution of high level programming languages. So the code might be portable for a few years, but to add longevity a more abstract representation (like algorithm diagrams) should also be included.

[1] `networkx.lanl.gov`

[2] `pythonware.com/products/pil/`

[3] `numpy.scipy.org`

5 SAGE source code

Most of the data in the code is stored in lists. This is a basic data structure in Python and it was chosen, because it is the simplest to use, although very far from optimum regarding usage of system resources (memory and processing cycles).

```
def int2list (number, radix, length) :
  list = Integer(number).digits(radix)
  return list + (length-len(list))*[0]

def list2int (list, radix) :
  return sum ( [radix^i*list[i] for i in xrange(len(list))] )

def list2bool (list) :
  return [Integer(list[i]>0) for i in xrange(len(list))]


class CA1D_rule () :
  def __init__(ca, k, m, r):
    ca.k = k; ca.m = m; ca.r = r;
    ca.f = int2list (ca.r, ca.k, ca.k^ca.m)
    ca.D = [zero_matrix(ZZ, ca.k^(ca.m-1), sparse=True) for k in xrange(ca.k)]
    for n in xrange(ca.k^ca.m) :
      o_l = n // ca.k; o_r = n % (ca.k^(ca.m-1))
      ca.D [ca.f[n]] [o_l, o_r] = 1
    ca.Sf = [ [ list2int ( list2bool (                                        \
                vector(int2list(i, ca.k, ca.k^(ca.m-1))) * ca.D[c]), ca.k)   \
                for i in xrange(2^(ca.k^(ca.m-1))) ] for c in xrange(ca.k) ]

  def __repr__(ca):
    return "Cellular Automaton:\n"+     \
      "  states    = "+str(ca.k)+"\n" + \
      "  neighbors = "+str(ca.m)+"\n" + \
      "  rule      = "+str(ca.r)

class CA1D_lattice () :
  def __init__(lt, ca, C, sh=0, b='cyclic') :
    lt.ca = ca; lt.C = C; lt.sh = sh; lt.N = len(lt.C); lt.b = b

  def __repr__(lt):
    return repr(lt.C)

  def next (lt) :
    if (lt.b == 'cyclic') :
      lt.C = [ lt.ca.f[list2int([lt.C[(x+lt.ca.m-1-i-lt.sh) % lt.N] \
               for i in xrange(lt.ca.m)], lt.ca.k)]                 \
               for x in xrange(lt.N            ) ]
    else :
      lt.C = [ lt.ca.f[list2int([lt.C[ x+lt.ca.m-1-i              ] \
               for i in xrange(lt.ca.m)], lt.ca.k)]                 \
               for x in xrange(lt.N-(lt.ca.m-1)) ]
      lt.N = lt.N - (lt.ca.m-1)
    return

  def prev (lt) :
    C_p = []

    if (lt.b == 'cyclic') :
      lt.D_x_b = [identity_matrix(ZZ, lt.ca.k^(lt.ca.m-1), sparse=True)]
      for x in xrange(lt.N) :
        lt.D_x_b.append (lt.ca.D[lt.C[lt.N-1-x]] * lt.D_x_b[x])
      lt.p = sum ([lt.D_x_b [lt.N] [i,i] for i in xrange(lt.ca.k^(lt.ca.m-1))])

      C_p = [CA1D_lattice(lt.ca, lt.N*[0], lt.sh, lt.b) for i in xrange(lt.p)]
      o_p0 = [];
    for o in xrange(lt.ca.k^(lt.ca.m-1)) :
      o_p0.extend([o for d in xrange(lt.D_x_b[lt.N][o,o])])
    o_p = list(o_p0)

    for x in xrange(lt.N) :
      i = 0
      while (i<lt.p) :
        o_L = o_p[i]; o_R = o_p0[i]
        for c in xrange(lt.ca.k) :
          n = o_L * lt.ca.k + c
          if (lt.C[x] == lt.ca.f[n]) :
            o_x = n % (lt.ca.k^(lt.ca.m-1))
            p_i = lt.D_x_b[lt.N-x-1][o_x,o_R]
            for p_c in xrange(p_i) :
              C_p[i].C [(x+lt.sh) % lt.N] = c
              o_p[i] = o_x
              i = i+1

  else :
    if (lt.b == 'open') :
      b_L = b_R = vector(lt.ca.k^(lt.ca.m-1)*[1])
      lt.b_x_b = [b_R]
    else :
      b_L = vector(lt.b[0]); b_R = vector(lt.b[1])
      lt.b_x_b = [b_R]

    for x in xrange(lt.N) :
      lt.b_x_b.append (lt.ca.D[lt.C[lt.N-1-x]] * lt.b_x_b[x])
    lt.p = b_L * lt.b_x_b[lt.N-1]

    C_p = [ CA1D_lattice(lt.ca, (lt.N+ca.m-1)*[0], lt.sh, lt.b)
            for i in xrange(lt.p) ]
    o_p = [];
    for o in xrange(lt.ca.k^(lt.ca.m-1)) :
      o_p.extend([o for d in xrange(b_L[o] * lt.b_x_b[lt.N][o])])
    for i in xrange(lt.p) :
      C_p[i].C [0:lt.ca.m-1] = int2list(o_p[i], lt.ca.k, lt.ca.m-1)

    for x in xrange(lt.N) :
      i = 0
      while (i<lt.p) :
        o_L = o_p[i];
        for c in xrange(lt.ca.k) :
          n = o_L * lt.ca.k + c
          if (lt.C[x] == lt.ca.f[n]) :
            o_x = n % (lt.ca.k^(lt.ca.m-1))
            p_i = lt.b_x_b[lt.N-x-1][o_x]
            for p_c in xrange(p_i) :
              C_p[i].C [x+lt.ca.m-1] = c
              o_p[i] = o_x
              i = i+1

  return C_p

def isGoE (lt) :
  if (lt.b == 'open') :
    s = 2^(ca.k^(ca.m-1))-1
    for x in xrange(lt.N) : s = lt.ca.Sf[lt.C[x]][s]
    return (s == 0)
  else :
    return "Unsupported boundary"
```

References

[1] Wuensche, A., Lesser, M.: The Global Dynamics of Cellular Automata. Addison-Wesley (1992)

http://www.cogs.susx.ac.uk/users/andywu/gdca.html

[2] Mora, J. C. S. T., Juárez, G., McIntosh, H. V.: Calculating ancestors in one-dimensional cellular automata. International Journal of Modern Physics C **15** (2004) 1151-1169

[3] McIntosh, H. V.: Linear Cellular Automata Via de Bruijn Diagrams (1994) http://delta.cs.cinvestav.mx/~mcintosh/newweb/marcodebruijn.html

[4] McIntosh, H. V.: Ancestors: Commentaries on The Global Dynamics of Cellular Automata by Andrew Wuensche and Mike Lesser (1993) http://delta.cs.cinvestav.mx/~mcintosh/oldweb/wandl/wandl.html

[5] Jeras, I., Dobnikar, A.: Algorithms for Computing Preimages of Cellular Automata Configurations. Physica D: Nonlinear Phenomena **233/2** (2006) 95-111 http://www.rattus.info/al/al.html

.

Cellular automata with cell clustering

Lynette van Zijl[*] and Eugene Smal

Computer Science, Stellenbosch University,
P/Bag X1, Matieland, 7601, South Africa
http://www.cs.sun.ac.za/~esmal

Abstract. We consider the modelling of a particular layout optimisation problem with cellular automata, namely, the LEGO construction problem. We show that this problem can be modelled easily with cellular automata, provided that cells are considered as clusters which can merge or split during each time step of the evolution of the cellular automaton. The LEGO construction problem has previously been solved with optimisation techniques based on simulated annealing and with a beam search approach, but we show that the use of cellular automata gives comparable results in general, and improves the results in many respects.

1 Introduction

Cellular automata (CA) and their many variations typically have a constant grid size, irrespective of the problem begin modelled. That is, in modelling a given problem using CA, the starting point is a grid of a fixed size consisting of single cells, where each cell has a specific interpretation. Each cell is evaluated simultaneously in each time step. However, in problems where *clustering* plays a meaningful role in the information represented by a cell, each *cluster* has to be evaluated at each time step in an efficient manner, instead of the traditional cell-by-cell evaluation. The LEGO construction problem falls into this last category. The LEGO construction problem, in short, concerns the optimal layout of a set of LEGO bricks to represent a given shape. In this article we discuss the modelling of the LEGO construction problem using CA, with emphasis on the necessity for efficient cluster evaluation. To our knowledge, the idea of CA with cell clustering has not been investigated before.

The rest of this article is organised as follows: In Sect. 2 we define the necessary terminology. In Sect. 3 we show how to encode the LEGO construction problem with cellular automata, followed by a discussion of the results in Sect. 4. We point out possibilities for future work in Sect. 5, and conclude in Sect. 6.

[*] This research was supported by NRF grant number 2053436.

2 Background

We assume that the reader is familiar with the theory of CA, as for example in [1]. We briefly recap on the definitions used in the rest of this article.

A cellular automaton C is a multidimensional array of automata c_i, where the individual automata c_i execute in parallel in fixed time steps. The individual automata c_i can reference the other automata by means of a *rule*. Typically, each c_i only references the automata in its immediate vicinity, called its neighbourhood.

If all the automata c_i are identical, then C is called a uniform CA. If all the automata c_i only have two possible states, then C is called a binary CA. We restrict ourselves to two-dimensional (2D) uniform binary CA in this article. When a given CA is two-dimensional, it forms a two-dimensional grid of cells, where each cell contains one of the automata c_i. In an $n \times n$ grid we assume that both the rows and columns are numbered from 0 to $n-1$.

As an example, consider the CA C with a 3×3 grid and with the rule $c_{ij}(t+1) = c_{i-1,j}(t) \oplus c_{i+1,j}(t)$. Suppose the rows of C are initialised with the values 000, 010 and 111 at time step $t = 0$ (see Fig. 1). Then the value of $c_{11}(t=1) = 0 \oplus 0 = 0$ and $c_{21}(t=1) = 1 \oplus 1 = 0$. Note that we assume null boundaries, so that if $i = 0$, then $i-1$ is simply ignored in the formula. Likewise, if $i = n-1$, then $i+1$ is ignored. In the example above, $c_{22}(t=1) = c_{12}(t=0) \oplus c_{32}(t=0) = c_{12}(t=0) = 1$.

$t = 0$

0	0	0
0	1	0
1	1	1

$t = 1$

0	0	0
1	0	1
1	0	1

Fig. 1. An example CA.

We now define a so-called *cluster*, which we will need to intuitively describe the LEGO construction problem as a CA. Usually, a CA has a given number of cells, and the number of cells stays fixed throughout all of its time evolutions. In our particular model, however, it is easier to assume that cells may merge or split during time steps, so that the number of cells in the CA may vary between different time steps.

We first define the *adjacency* of cells to mean cells that touch on a joint border:

Definition 2.1. *Let C be a 2D CA. Two cells c_{ij} and c_{km} in C are adjacent if either $|i-k| = 1$ or $|j-m| = 1$. If both $|i-k| = 1$ and $|j-m| = 1$, then the cells are not adjacent.*

We now define a cluster as a set of adjacent cells:

Definition 2.2. *Let C be a 2D CA. A cluster $\mathcal{B}$ in C is a set of cells from C such that any cell c_{ij} in $\mathcal{B}$ is adjacent to at least one other cell c_{km} in $\mathcal{B}$.*

It is convenient to define disjointed clusters:

Definition 2.3. *Two clusters $\mathcal{A}$ and $\mathcal{B}$ are disjoint if there is no cell a_{ij} in $\mathcal{A}$ such that a_{ij} is also in $\mathcal{B}$, and no cell b_{ij} in $\mathcal{B}$ such that b_{ij} is in $\mathcal{A}$.*

The clusters in our modelling of the LEGO construction problem are typically disjoint, but it is not a necessary condition for our algorithms to function correctly.

We can also define adjacency of clusters:

Definition 2.4. *Two clusters $\mathcal{A}$ and $\mathcal{B}$ are adjacent if there exists at least one cell a_{ij} in $\mathcal{A}$ and at least one cell b_{km} in $\mathcal{B}$ such that a_{ij} is adjacent to b_{km}.*

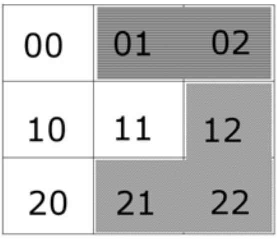


Fig. 2. Cell clustering in a CA.

Example 2.1. In Fig. 2, cell c_{11} is adjacent to cell c_{12}, but c_{11} is not adjacent to c_{22}. Also, the set of cells $\{c_{11}, c_{21}, c_{22}\}$ forms a cluster, but the set $\{c_{12}, c_{21}\}$ does not. Lastly, the clusters $\{c_{01}, c_{02}\}$ and $\{c_{12}, c_{21}, c_{22}\}$ are adjacent, since cells c_{02} and c_{12} are adjacent. These two clusters are also disjoint.

□

When using clusters to model the LEGO construction problem, we will further restrict the cluster to have only the forms that represent standard LEGO bricks. That is, the clusters can only have the rectangular shapes and sizes as shown in Fig. 3. Note how the bricks are identified by the number and rectangular arrangement of its studs. For example, a brick with two rows of three studs each is called a 2 × 3 brick.

Definition 2.5. *A valid LEGO brick is one with dimensions as defined in Fig. 3.*

The reader should note that clusters are considered to be homogeneous entities, in the sense that all the cells forming the cluster will contain the same status information. However, clusters are typically of different sizes (for example, the cluster representing a 2 × 4 LEGO brick may lie next to the cluster representing a 1 × 3 LEGO brick). This means that we have to define the meaning of the neighbourhood of a cluster.

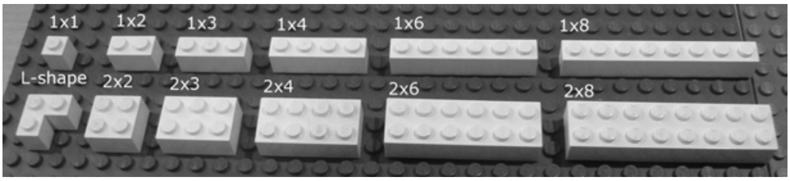


Fig. 3. The list of standard LEGO bricks.

Definition 2.6. *The (Von Neumann) neighbourhood of a cluster $\mathcal{A}$ is the collection of clusters adjacent to $\mathcal{A}$.*

Note that the above definition implies that a cluster may not necessarily have four neighbours in its Von Neumann neighbourhood. Also, the number of Von Neumann neighbours may vary between different clusters. For example, in Fig. 2, suppose that the there are four 1×1 clusters, namely, are c_{00}, c_{10}, c_{20} and c_{11}. Also, $\{c_{01}, c_{02}\}$ forms a cluster of size 1×2 and $\{c_{12}, c_{21}, c_{22}\}$ forms an L-shaped cluster. Then the Von Neumann neighbourhood of cluster c_{11} consists of three other clusters, namely, the cluster c_{10}, the cluster $\{c_{01}, c_{02}\}$ and the cluster $\{c_{12}, c_{21}, c_{22}\}$.

In addition, a single cluster may have a different number of neighbours in different time steps. We also note that the definition of different types of neighbourhoods (such as Von Neumann or Moore) now simply defines the type of adjacency. This is indeed the case with CA without cell clustering as well, but the neighbourhoods based on a fixed size grid enforce a fixed number of neighbours for all cells through all time steps.

The next issue to consider is the merging of clusters.

Definition 2.7. *Let $\mathcal{A}$ and $\mathcal{B}$ be two clusters in a binary 2D CA. Define a new cluster $\mathcal{C}$ to contain every cell a_{ij} in $\mathcal{A}$ and every cell b_{km} in $\mathcal{B}$, and no other cells. Then $\mathcal{C}$ is said to be the merge of $\mathcal{A}$ and $\mathcal{B}$.*

Given a merge operation on two clusters, it is natural to consider the dual operation of splitting up a cluster into smaller parts. In our case, we simply consider the splitting of a cluster into its smallest constituent parts, namely, clusters of size 1×1:

Definition 2.8. *Let $\mathcal{A}$ be a cluster in a binary 2D CA, and let $\mathcal{A}$ contain k distinct cells c_i, with $1 \leq i \leq k$. Then the operation split($\mathcal{A}$) creates k new clusters $\mathcal{A}_i$, such that c_i is the only cell in $\mathcal{A}_i$, for $1 \leq i \leq k$. All the clusters $\mathcal{A}_i$ are thus disjoint.*

We now discuss the LEGO construction problem in more detail, and show how to encode it using a uniform binary CA with clustering.

3 Encoding the LEGO construction problem as a CA

The LEGO construction problem in essence concerns the following issue: given a real-world three-dimensional (3D) object, find the most optimal way to build a LEGO sculpture of the object, given a specific set of standard LEGO bricks.

The traditional approach to solving the LEGO construction problem is to virtually cut the 3D object into horizontal two-dimensional (2D) layers (we assume that a digital representation of the real-world 3D object is given as input). The problem then reduces to a series of 2D solutions which can be joined together (under certain restrictions) to find the final 3D solution. We note that each horizontal layer will eventually form one layer of bricks in the final solution.

Given the digital representation of a 2D layer, the layer is then encoded as a grid of zeroes and ones, where the ones represent the solid parts of the original object in this layer. To solve the LEGO construction problem, we therefore have to find an optimal layout of LEGO bricks to cover the ones in each layer. It is assumed that there is an unlimited number of each type of brick of the standard LEGO bricks.

An optimal layout of LEGO bricks is defined by a number of factors [2], the most important of which are:

- the number of bricks used should be minimised;
- larger bricks should be used where possible, to allow for ease of building, better brick connectivity and to indirectly reduce the number of bricks used;
- bricks in consecutive layers should have alternate directionality to increase the stability of the final sculpture;
- a large part of each brick in the current layer should be covered by other bricks in the next layer; and
- a brick in the current layer should cover as many other bricks in the previous layer as possible.

The traditional way of solving the LEGO construction problem is to use combinatorial optimisation methods to find an optimal layout [3]. One of the first proposed solutions was based on genetic algorithms [4], while the current *de facto* solution makes use of a beam search to cover the search space [5]. As with all optimisation methods, some models can take excessively long to solve. Also, in some cases, sub-optimal solutions are found when the search tree is trimmed. Our CA approach is an attempt to alleviate exactly these problems.

We are now ready to encode the LEGO construction problem as a CA. Assume a two-dimensional grid of clusters, representing one 2D layer of the real-world object as described previously. Each 1×1 cluster has the value zero or one, which indicates whether this is an area to be filled with a brick, or to be kept open. Additionally, we allow each cluster to keep separate status information as needed.

The time evolution of each layer of the model intuitively proceeds as follows: initially, each cluster of size 1×1 which contains a 1 value, is assumed to represent a LEGO brick of size 1×1. Then, for each time step, there are two phases.

In the first phase, each cluster investigates every other cluster in its Von Neumann neighbourhood, to decide whether it is *possible* to merge with that neighbouring cluster. The status information for each cluster is then updated to indicate with which of the clusters in its neighbourhood the cluster is willing to merge. This operation happens in parallel and simultaneously for all clusters, similar to the time evolution of cells in a standard CA.

In the second phase, a sequential pass through the individual clusters investigates the status information of each cluster. This status information is used to decide which clusters will merge to form larger clusters. As the merging into clusters takes place in a sequential and progressive fashion, the order of the actual merges can be random, or front-to-back, or any other implementable way of choosing the order. This has a slight influence on the final layout.

We now discuss each phase in more detail.

3.1 Merging

In the first phase of the merging step, each cluster examines its neighbours to determine with which of the neighbours it can merge to form a valid new LEGO brick. We assume a Von Neumann neighbourhood, which uses the clusters directly adjacent to the current cluster. If there is more than one possible merge amongst the neighbours, the cluster selects the best possible merge by using a local cost function rule (described below)[1]. If there are no clusters with which the current cluster can merge, the cluster can split into smaller clusters (see Sect. 3.2). The status information of the cluster is now updated to indicate its best possible merge neighbour.

We use a local cost function as the rule to determine with which neighbour each cluster should be merged. The local cost function is defined as:

$$\begin{aligned} Cost = {} & W_{perpend} \times perpend \\ & + W_{edge} \times edge + W_{uncovered} \times uncovered \\ & + W_{otherbricks} \times otherbricks \end{aligned}$$

where the W's are weight constants and

- the *perpend* variable corresponds to the directionality of the bricks in consecutive layers;
- the *edge* variable represents the number of the edges of each brick that coincide with edges of bricks from the previous layer;
- the *uncovered* variable describes how much of the area of each brick is not covered by bricks in the previous and following layers; and
- the *otherbricks* variable represents the number of bricks in the previous layer covered by this brick.

[1] If the local cost function evaluates to the same minimal value for more than one neighbour, a random choice is made amongst the minimal value neighbours.

This cost function directly corresponds to the fitness function used by Petrovic [4], but without the *numbricks* and *neighbour* heuristics used in his function. These were removed since they have no effect when local optimisation is used. The *numbricks* heuristic is used to minimise the total number of bricks globally. Our method indirectly minimises the number of bricks, since a brick will always merge with a neighbouring brick if possible. Even though the *numbricks* heuristic was not included in the local cost function, we do still add it in our test results when calculating the total sculpture cost. This is useful in comparing the results to those achieved with other methods.

The cost function must be minimised to help ensure that the total cost of the LEGO sculpture is kept to a minimum while its stability is maximised. After each time step, the total cost for the layer is calculated. The best layer constructed throughout the time steps is used as the final building instruction for that layer.

In the second phase, the new set of (merged) clusters are constructed. The process involves two steps:

1. Each cluster is considered, and a potential new cluster is calculated by using the information from phase 1. If the new cluster forms a valid LEGO brick, the merge information for the cluster is completed.
2. For all potential new clusters that do not form valid bricks, the potential new cluster is traversed until a further traversal step would lead to an invalid brick. The traversed clusters are then flagged for merging into a new cluster, and the process is repeated until all the elements of the potential cluster have been traversed and flagged for merging into valid brick clusters.

Clearly, phase 2 is a sequential phase, where all clusters are calculated sequentially by visiting each cluster and using its best possible merge status information.

Once the new clusters have been calculated, all the clusters are merged simultaneously. The exact detail for the implementation of the merging can differ substantially, and this can have a noticeable effect on the performance of the method. In our case, we simply merge the clusters by starting with the first cluster in the new potential cluster, and merging it with its best possible merge neighbour. The resulting cluster is then merged again in a similar manner until either it cannot merge any further, or the whole cluster has been merged together. We note that this method of merging the clusters will not necessarily always deliver the optimal result. However, the local optimisation propagates to a satisfactory result in almost all cases, with a low execution time.

We give an example of the merging process below.

Example 3.1. Suppose that a 3×3 grid, with all values consisting of ones, must be constructed from the standard LEGO brick set (see Fig. 4). Therefore, there are initially nine clusters in the grid, each representing a brick of size 1×1.

In the first phase each cluster determines with which of its neighbours it can merge to form a new valid brick. The cluster then sets its status information

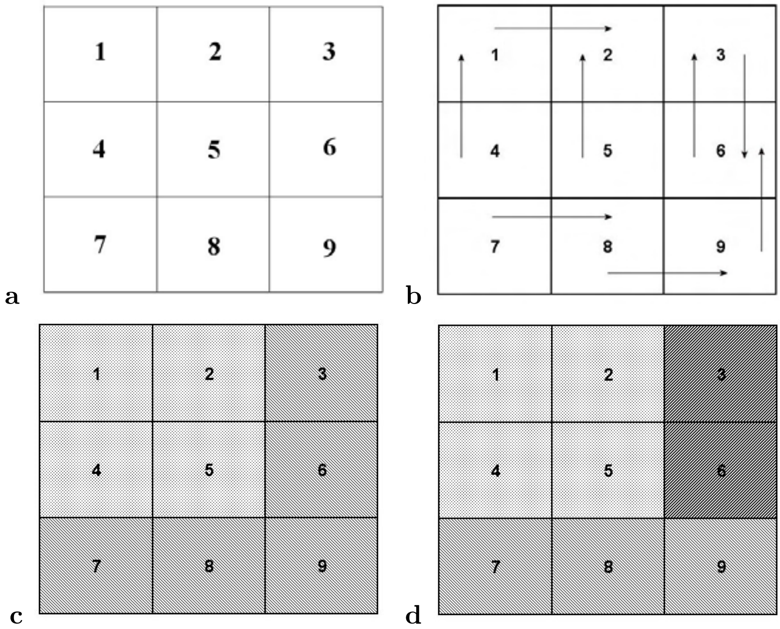


Fig. 4. (a) The 2D grid, (b) potential merge neighbours, (c) potential new clusters, (d) the final three clusters.

to indicate with which neighbouring brick it would best merge (see Fig. 4(b)). Here, each cluster indicates its best possible merge brick with an arrow.

In the second phase, all the potential new clusters are calculated. In this example we have two clusters (see Fig. 4(c)).

After the potential new clusters are calculated, they are traversed and merged together to form the larger clusters. The merging process in this case ends with three clusters instead of two, since the second cluster cannot merge into one large valid brick (see Fig. 4(d)). The second cluster is therefore merged, until it cannot merge any further without resulting in an invalid brick. The remaining clusters are then traversed and merged together to form the third cluster.

In the next time step (see Fig. 5), each cluster will again determine with which brick its group of clusters should merge, using its local neighbourhood, to form a new valid brick. The best possible merge for the group is then selected. In this example cluster $\{1, 2, 4, 5\}$ can only merge with cluster $\{3, 6\}$ to form a valid larger brick. Cluster $\{7, 8, 9\}$ can then not merge with any of the other two bricks. Therefore, there will only be one cluster to merge containing the merge of $\{1, 2, 4, 5\}$ and $\{3, 6\}$. After the clusters have been merged, there are two bricks in the layout, which cannot be merged again into a larger valid brick.

□

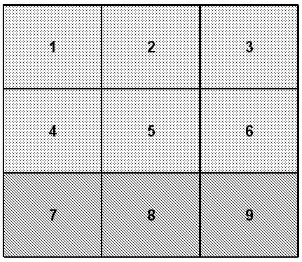


Fig. 5. The resulting bricks after time step 2.

The counterpart to the merge operation is the *split* operation. As our merge operation can quickly reach the point where no more merging is possible, it is useful to split a given cluster so that the search for an optimal solution can continue.

3.2 Splitting

Our splitting method simply splits a given cluster into 1×1 clusters (see Fig. 6). It is in principle possible to use other splitting strategies (such as splitting into k smaller clusters). However, due to the large number of possible brick sizes in the LEGO construction problem, this would be computationally expensive with no immediate advantage over our complete dissolution of the cluster.

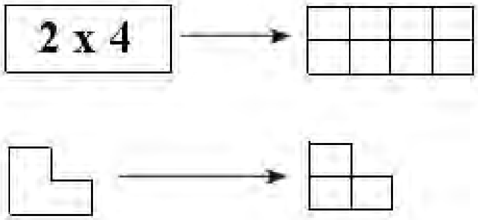


Fig. 6. Two bricks splitting into 1×1 bricks

A cluster can potentially split if it cannot merge with any of its neighbouring clusters. We assign a splitting time delay to each cluster, which is the maximum number of time steps during which the cluster will unsuccessfully attempt possible merges. When the splitting time delay expires, the cluster will attempt to split. The time delay allows neighbouring clusters enough time to grow into clusters with which the current cluster can merge. If there were no time delay, clusters could split too early and larger clusters will fail to form. The time delay can be a random or fixed number of time steps for each cluster.

If a cluster was unable to merge within the time delay, it will attempt to split. For the LEGO construction problem, we assign a different splitting probability to every brick (basically, the larger a brick, the less its splitting probability). When a cluster attempts to split, it generates a random number. If that number is less than its splitting probability, the cluster splits. Otherwise, the time delay of the cluster is reset and the cluster will again try to merge with neighbouring clusters.

In summary, our algorithm considers each layer separately. For each layer, it executes phase 1 and phase 2, and calculates the cost for the layer. Phase 1 finds the best merge neighbour for each cluster, and phase 2 calculates new clusters sequentially before merging all the clusters in parallel.

We implemented our CA-based method, as well as the beam search method of Winkler [5]. In Appendix A, we give some screenshots of our system, as well as some of the LEGO instructions generated by the system. In the next section, we analyse the results obtained by our CA with cell clustering method.

4 Results

We implemented our CA with cell clustering method, as well as the beam search method. We compare these two methods below, and discuss their respective advantages and disadvantages. We also point out some interesting issues that occurred in the implementations.

In comparing the CA with cell clustering against the beam search method, one is in essence comparing a local optimisation method against a global optimisation method. Having noted that, we had to define measures of quality for the comparison. We list these below:

- *number of bricks used in the final model*: in general, the fewer bricks used, the cheaper it is to produce the model. This was one of the criteria listed by the LEGO company in the original definition of the problem;
- *cost (as defined by the cost function)*: as the cost function contains all the parameters against which to optimise, a low cost function represents a ‘good’ model;
- *execution time of the implementation on a specific model*: the quicker a good layout can be reached, the better;
- *extending the solution to allow coloured sculptures*: the traditional solutions to the LEGO construction problem assume the input to be monochrome, which significantly limits the practical use of the implementation;
- *ease of building*: this is a ‘fuzzy’ measure, and we simply experimented by building a number of the same sculptures based on the instructions generated by the beam search method and the instruction generated by the CA method, respectively; and
- *ease of implementation of the solution*: here, we consider the complexity of the coding of the solution.

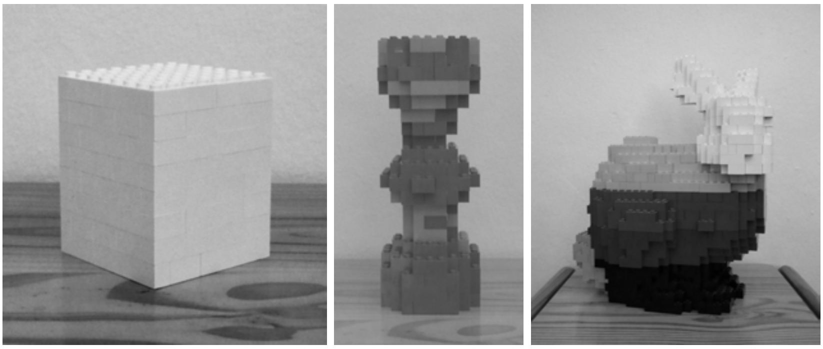


Fig. 7. The LEGO sculptures.

We set up a number of experiments to evaluate our measures of quality. For the experiments, we selected three different models (see Fig. 7). The first is a simple hollow cube with dimensions 8 × 8 × 8 unit bricks, which is representative of small geometric models without rounded surfaces. The second is a chess pawn with dimensions 10 × 10 × 20 unit bricks, which is representative of small models with many rounded surfaces. The third is the well-known Stanford bunny with dimensions 30 × 20 × 25 unit bricks, which is representative of a larger sculpture with more complexity and finer detail. We selected weight constants [2] $C_{numbricks} = 200$, $C_{perpend} = -200$, $C_{edge} = 300$, $C_{uncovered} = 5000$, and $C_{otherbricks} = -200$.

Reaching a good layout with the CA method depends on the number of time steps that the CA executes, whereas to reach a good layout with the beam search method depends on the width of its search tree (and how much it is pruned). These two parameters are not necessarily easily comparable. In addition, there is a random element to be taken into account in the CA method (in the merging phase), so that the values given for the CA method are averaged over a number of runs.

A summarised view of the results is given in Table 1. Here, the number of iterations for the CA was set at 1500, and the width of the search tree in the beam search method was set to 4000 (a wider search tree is possible, but will dramatically increase the execution time of the beam search).

The first measure is the number of bricks in the final sculpture. From Table 1 it is easy to see that, for the more complicated sculpture (the chess pawn) and the larger sculpture (the bunny), the CA method uses less bricks than the beam search method. This is to be expected, as the CA method forces merging into larger bricks whenever possible so that fewer but larger bricks are used. The beam search, on the other hand, will evaluate the size of the bricks as part of its cost function, and the requirements for non-coincidental edges and alternate

[2] These values are similar to the values suggested by [4]. The large integer values prevent underflow errors in the evaluation of the cost function.

Model	Number of bricks		Cost function value		Execution time (s)	
	CA	BS	CA	BS	CA	BS
Cube	42	43	20357	20158	75	8
Chess pawn	170	196	99889	106543	181	18
Bunny	1133	1436	630739	714223	280	313

Table 1. Experimental results for CA with cell clustering (CA) and beam search (BS).

directionality then result in smaller bricks being chosen. It is possible to carefully choose the weight constants for the beam search to weigh the size of the bricks as highest priority. However, that leads to weaker sculptures, as the lack of alternate directionality and non-coincidental edges result in disconnected sculptures. Therefore, in the beam search, it often happens that two adjacent smaller bricks rather than a single larger brick appears. For example, we often found two adjacent 1 × 3 bricks produced in the beam search sculptures, whereas the CA method would almost always force those into a single 2×3 brick. This influences the ease with which a sculpture can be built, as it is usually more uncomfortable to connect many small bricks than to connect a single larger brick. In that sense, the CA method produces 'better' building instructions.

However, the above comments on the number and size of the bricks are mostly true for larger and more complex sculptures. In simple and small sculptures, the beam search can stabilise more quickly than the CA approach. For example, for the hollow cube, if the tree width is set at 10000, the number of bricks for the final sculpture is reduced to 32 within 19 seconds. The CA, on the other hand, takes 2000 time steps and 96 seconds to reach a brick count of 40.

Our second measure of quality is the value of the cost function. This represents the quality of the sculpture in total, including number of bricks, alternate directionality and non-coincidental edges between layers. Here, as can be seen in Table 1, the difference between the CA method and the beam search is comparable. It should be noted that it is possible to get a smaller cost value for the beam search, but at the expense of a much wider search tree and substantial increase in execution time. For example, if we increase the width of the tree to 40000 for the chess pawn, the value of the cost function falls to 95366, but the execution time increases to 275 seconds. Corresponding to the values in Table 1, the sculptures built from the CA generated instructions and the beam search generated instructions showed no perceivable difference in stability.

The last direct measure of quality of our method is the execution time. Although Table 1 indicates an order of magnitude difference in the execution times between the two methods, the values can be potentially misleading. We note that the execution of the CA is dependant on the number of time steps, which is a user parameter. There is no easy way to let the implementation decide its own point where no improvement is possible. For example, with the chess pawn model, a number of 192 bricks is reached within 14 seconds and just 100 iterations. To improve the number of bricks to 168 takes 2000 iterations and 246 seconds. The beam search, on the other hand, could reduce the number of bricks only to 180 in 275 seconds (with a tree width of 40000). In addition, the CA has a transitional startup phase before stabilisation which shows clearly in the smaller models, but which is relatively shorter in the larger models. However, it is fair to say that for small and simple models, the beam search method executes faster than the CA method on a single processor machine. As we point out in the next section, we intend to parallelise both methods in the future.

There are two areas where the CA method shows distinct advantages over the beam search method. The first is ease of implementation – although the number of lines of code for each of our implementations is roughly comparable, it was much simpler to implement the CA (even with merging and splitting) than it was to implement a search tree with its associated pruning. The second area where the CA has a distinct advantage over beam search, is the extension of the implementation to cater for coloured sculptures. In the case of the CA method, this has a small effect on the merging phase, where merging of clusters is restricted to clusters of the same colour only. Our current implementation of the CA method already incorporates coloured models. The beam search method, however, is much more difficult to extend to coloured models. Every sculpture has a certain 'thickness' – that is, the depth of any one side of the sculpture. Traditionally, a thickness of four unit bricks works well to ensure enough stability without unnecessarily increasing the number of bricks used. When colour is incorporated, it is possible to make the outer brick the required colour and use the inner bricks (that are hidden from view) to ensure stability. In that way, even checkered patterned sculptures can be built without losing stability. However, in the beam search, this single extra requirement leads to an explosion in the size of the search tree, and in many cases tree pruning leads to disconnected sculptures.

Our last observation is a rather obvious one, but still important: the CA method uses significantly less memory than the beam search method.

In summary then, we conclude that the CA method shows results comparable to that of the beam search method, except for small and simple models. For complex models, the CA model typically uses fewer bricks. And lastly, the distinct advantages of the CA method is the ease of implementation and the fact that it is easily extendable to handle coloured sculptures.

5 Future work

It is not immediately obvious that a theoretical investigation into CA with cell clustering would yield interesting results. However, we plan to consider the concept of clustering in more detail, even if only for modelling some additional problems.

As far as our implementation is concerned, we plan to parallelise the code and run experiments on a Beowulf cluster to consider the extent of the speedup over a single processor machine. As the parallel implementation of game trees is well-known, we do not expect either method to improve upon the other by using parallelisation.

We have previously implemented a 3D CA with clustering, in order to do the layout for all layers in the sculpture simultaneously. The results were poorer than expected, but we intend to revisit this aspect in more detail.

Although not part of the CA solution to our problem, we wish to improve on the manual effort necessary to build a virtual 3D model for input to the system. Currently, the user must construct a 3D model by hand using a 3D modelling package. However, ideally, it should be possible to input a series of synchronised photographs, so that the system can build its own 3D model. We have completed an initial version of such a system based on the shape from silhouette technique [6], but there are still some issues to be resolved.

6 Conclusion

The paper presented a new approach to solving the LEGO construction problem, based on a CA with cell clustering.

We showed that a CA incorporating cell clustering is a simple generalisation of traditional CAs. This generalisation allows us to easily model real world clustering problems with CA.

We discussed the practical results from our implementation of the LEGO construction problem, and pointed out the differences between the traditional beam search approach and the CA approach. We pointed out the various circumstances in which one approach would perform better than the other.

Finally, we stated our planned extensions to our implementation, and noted the possibility for parallelising the implementation on multiple processors, as well as the need for an easier method of input creation.

References

[1] Wolfram, S.: Cellular Automata and Complexity. Perseus Books Group (2002)

[2] Na, S.: Optimization for layout problem. PhD thesis, University of Southern Denmark (2002) http://www.idi.ntnu.no/grupper/ai/eval/brick/.

[3] Gower, R., Heydtmann, A., Petersen, H.: LEGO: Automated model construction. 32nd European Study Group with Industry (1998) 81–94 http://www.idi.ntnu.no/grupper/ai/eval/brick/.

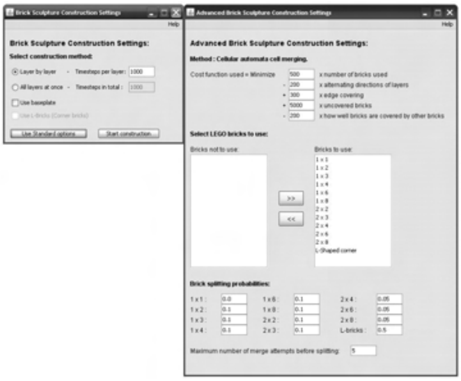
[4] Petrovic, P.: Solving the LEGO brick layout problem using evolutionary algorithms. Technical report, The Maersk Mc-Kinney Moller Institute for Production Technology, Norwegian University of Science and Technology (2001) http://www.idi.ntnu.no/grupper/ai/eval/brick/.

[5] Winkler, D.: Automated brick layout. BrickFest (2005) http://www.brickshelf.com/gallery/happyfrosh/BrickFest2005/.

[6] Cheung, K., Baker, S., Kanade, T.: Visual hull alignment and refinement across time: a 3D reconstruction algorithm combining shape-from-silhouette with stereo. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (June 2003) 375–382

Appendix A

This appendix contains some screenshots and results generated from our system.



Brick Sculpture Construction Settings
Help

Brick Sculpture Construction Settings:

Select construction method:

● Layer by layer - Timesteps per layer: 1000

○ All layers at once - Timesteps in total : 1000

□ Use baseplate

□ Use L-Bricks (Corner bricks)

Use Standard options
Start construction
Advanced Brick Sculpture Construction Settings
Help

Advanced Brick Sculpture Construction Settings:

Method : Cellular automata cell merging.

Cost function used = Minimize 500 x number of bricks used
- 200 x alternating directions of layers
+ 300 x edge covering
+ 5000 x uncovered bricks
- 200 x how well bricks are covered by other bricks

Select LEGO bricks to use:

Bricks not to use:

Bricks to use:
1 x 1
1 x 2
1 x 3
1 x 4
1 x 6
1 x 8
2 x 2
2 x 3
2 x 4
2 x 6
2 x 8
L-Shaped corner

Brick splitting probabilities:

1 x 1 :	0.0	1 x 6 :	0.1	2 x 4 :	0.05
1 x 2 :	0.1	1 x 8 :	0.1	2 x 6 :	0.05
1 x 3 :	0.1	2 x 2 :	0.1	2 x 8 :	0.05
1 x 4 :	0.1	2 x 3 :	0.1	L-bricks :	0.5

Maximum number of merge attempts before splitting: 5

Fig. 8. Some parameter settings for the system.

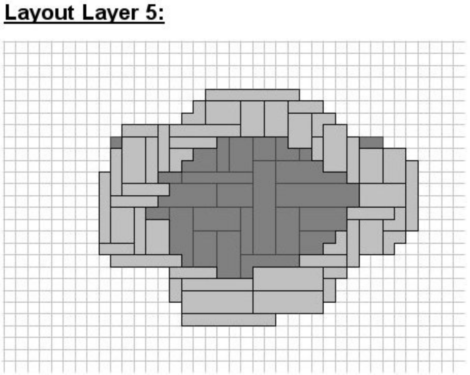


Fig. 9. Instructions generated for the 5th layer of the Stanford bunny. The dark grey areas indicate bricks in previous layers, and the light grey the bricks of the current layer.

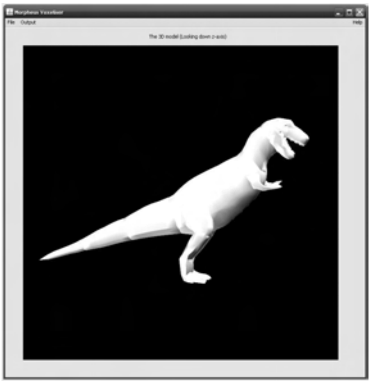
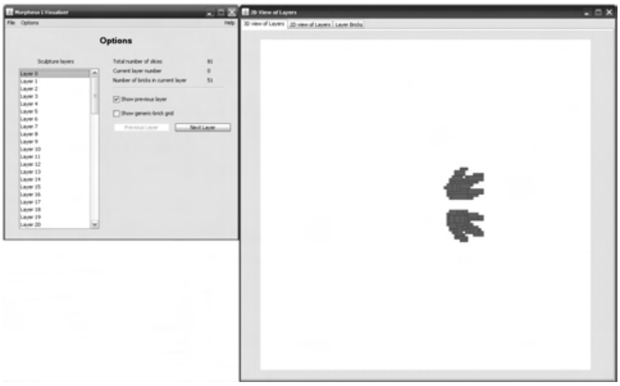


Fig. 10. A 3D model of a dinosaur.



Total number of slices 81
Current layer number 0
Number of bricks in current layer 51

☑ Show previous layer

☐ Show generic brick grid

Fig. 11. Instructions for the dinosaur being generated.

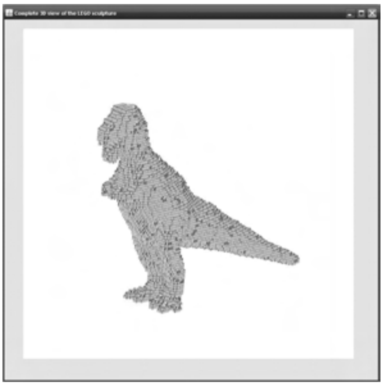


Fig. 12. Completed LEGO sculpture of the dinosaur.

From cellular automata to a random artificial chemistry producing symmetric replicas

Andre Barbé

Department of Electrical Engineering, Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B3001 Leuven, Belgium

1 Introduction

The algorithm that will be presented here, grew out of an earlier one that was reported in [1]. We briefly recall the original algorithm, and sketch some intermediate transition steps that lead to the artificial chemistry.

Consider the one-dimensional CA, with states in $\{0, 1\}$, and with local evolution rule (state transition rule)

$$c_{t+1}(k) = (c_t(k) + c_t(k+1)) \bmod 2 \tag{1}$$

where $c_t(k)$ refers to the state of cell at position k at (discrete) time t. When the initial state of the CA is only known over a finite string of cells, applying the rule creates an orbit which, when displayed properly, takes the shape of a configuration on an equilateral triangular cellular array (TCA), as illustrated in Fig. 1 A. Now consider the following

Random feedback algorithm

RF step 1 : Initialize the top row configuration of the TCA (i.e., assign states 0 or 1 to the cells of the top row, possibly in a random way)

RF step 2 : Complete the TCA-configuration by applying the local rule.

RF step 3 : Select randomly a number of cells on the right side of the TCA, and feed the state of these cells back to the corresponding top row cells (see Fig. 1 B). This produces a new top row which keeps the states of the cells which are not affected by the feedback unchanged, while states on the feedback-affected cells copy the states of the corresponding right side cells (and thus *may* change state).

RF step 4 : Return to step 2.

It was observed in [1] that iterating this algorithm eventually produces a TCA-configuration that is invariant under rotations of 0°, 120°, and 240°. I.e., the configuration has Z_3-symmetry. And this no matter the size of the TCA.

Moreover, it was observed that the random feedback may actually go from any side of the TCA to any other side (varying randomly from iteration to iteration). This is plausible by the Z_3-symmetry of the TCA and of the local matching rule: the TCA-configuration can be grown by applying the local matching rule

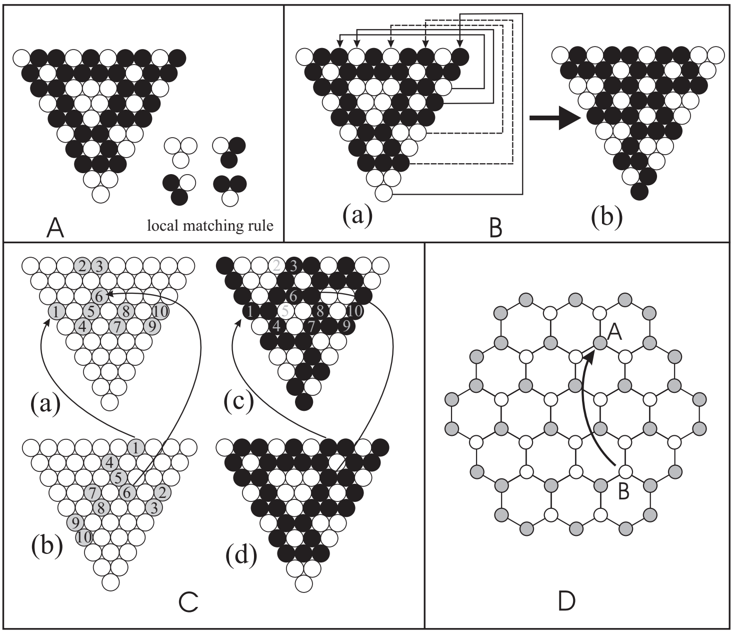


Fig. 1. A: A triangular cellular array (TCA)with state configuration pattern: it is the orbit of a one-dimensional CA when given the state configuration on the top side (state 0=white, state 1=black), and satisfying the rule (1). This induces the symmetric local matching rule as displayed. B: Implementing a random feedback: (a) the states of some randomly selected cells on the right side are fed back to the top side, as indicated by the connecting lines. Full lines induce an effective change of state on the top row, the dotted lines do not. (b) is the resulting configuration obtained by applying the local matching rule implementing the feedback-induced changes on the top side. This illustrates one iteration step in the random feedback algorithm. C: (a) a nontrivial generating set for the TCA is formed by the gray cells (grayness does not indicate a cell state): it is a minimal set of cells such that knowledge of the state on these cells completely determines the overall configuration under application of the local matching rule. (b) shows the clockwise rotation by 120° of this generating set, and the arrows between (b) and (a) denote the feedback lines between the two (randomly) selected cells 1 and 6 in the generating set (in analogy to the feedback lines between right side and top side in B). (d) displays a state configuration, and (c) shows how, after having implemented the feedback from the cells 1 and 6 (thereby leaving the states on the other cells of the generating set in (a) unchanged), the overall configuration changes. D: the gray cells form a (nontrivial) generating set on a D_6-symmetric frame cellular automaton (for a local rule stating that the states (in the binary field $\mathbb{F}_2$) of all cells in an elementary hexagon should add up to 0) . When cell A is a (randomly) selected cell from the generating set that could change its state under proper counterclockwise 120° feedback, it will change its state into the state of cell B.

from the configuration on any side. Unfortunately, the algorithm's dynamics could only be analyzed for TCA's of very limited size, and a general proof for this symmetric outcome is still lacking.

It was also shown that the random feedback need not necessarily have to take place between cells at the sides of the TCA. These sides are just the most natural examples of so-called *generating sets* for the TCA, i.e. minimal subsets of cells in the TCA such that knowledge of the states on these cells determines in a unique way the overall TCA-configuration (whereby of course the local matching rule induced by (1) has to be satisfied). The aforementioned feedback works equally well when replacing the top side by any generating set (and replacing the right side/left side by the corresponding 120° clockwise/counterclockwise rotation of this generating set). This is illustrated in Fig. 1 C. Further properties of generating sets in connection with (binary) quasigroup-defined TCA's, and extensions of TCA's known as frame cellular automata (involving multary quasigroup-induced matching rules, see Fig. 1 D for an example), can be found in [2], [3].

We will now show how to obtain a procedure that is completely equivalent with the feedback algorithm, but without using the explicit feedback. This is illustrated in Fig. 2. In step 3 of the algorithm, just duplicate the (master) TCA, and call the duplicate the slave. Then rotate the slave 120° counterclockwise, and let it align with the master TCA. Then the master copies, on the top side, randomly some of the cells from the top side of the rotated slave. After which, by going back to step 2, the master develops its full configuration by applying the local matching rule. This repeats iteration after iteration. It is clear that this procedure is equivalent with the original algorithm: the random feedback has been replaced by a random copying. Corresponding (aligned) cells on the top side of the master and on the top side of the rotated slave that have the same state remain in that state: some of the remaining cells in the master's top side may copy the state of the corresponding slave cells. This can be done with a certain probability.

As mentioned before, the original algorithm works equally well by changing the sides involved in the feedback in each iteration. In the master-slave copying procedure, this means that the slave can be rotated arbitrarily, and that the copying action of the master may be done along any side. Or, for the same purpose: along any generating set. The point is that master and slave may reorient relatively to each other by rotations which are a multiple of 120°. This opens a way to get rid of the hierarchical master-slave distinction. One could start with two TCA's with the same initial configuration (possibly generated randomly). Let them reorient randomly (by random rotations over multiples of 120°), after which they align and make a random copy of each other along a generating set. This means: where corresponding cells of the the two rotated TCA's on the generating set have the same state, they keep this state. Where corresponding cells differ: bring both cells in state 0 or in state 1, each with probability 1/2. This makes the configuration on the generating sets on both TCA's the same, and accordingly, the new overall configuration will also be the

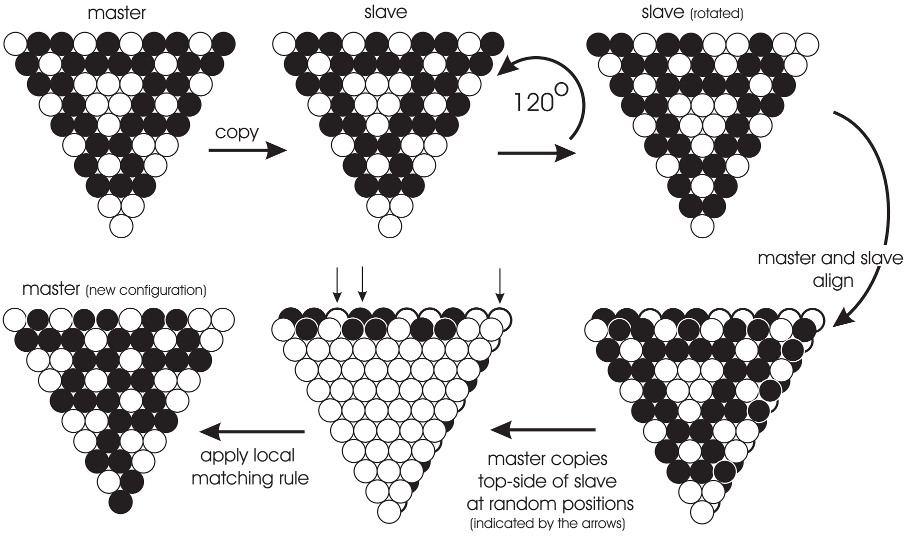

master
slave
slave (rotated)
copy
120°
master and slave align
master (new configuration)
apply local matching rule
master copies top-side of slave at random positions
(indicated by the arrows)

Fig. 2. One iteration cycle in the implementation of the random feedback scheme using a slave cellular automaton.

same. Then this procedure iterates, and eventually produces two TCA's with a Z_3-symmetric configuration. In that sense, there is no difference with the original feedback algorithm, and it brings also nothing new when looking for a proof of this observation. We have even performed similar experiments with Z_4- and Z_6-symmetric two-state frame cellular automata, in which the reorientations are randomly taken from the corresponding symmetry group (for example, Fig. 1D shows an automaton with Z_6-symmetric shape (Z_6 being a subgroup of D_6)). The observations are also similar: the configurations obtained are respectively Z_4- and Z_6-symmetric.

In the search for a proof, having to take into account a local matching rule, presents a difficulty. So we tried cellular arrays without local matching rule. This is the limiting case of a degenerate frame cellular automaton in which the generating set coincides with the whole set of cells in the automaton. At the same time, we tried a bold generalization: if the copying procedure sketched above works for two TCA's, maybe it will also work for more than two TCA's. And indeed, these generalizations, which in hindsight define an artificial chemistry, allow several observations to be proven for any size of underlying array (frame).

2 Description of the artificial chemistry

An artificial chemistry is a triple $(\mathcal{C}, D, I)$, where $\mathcal{C}$ is a collection of "artificial molecules", moving around under a dynamics D, and interacting according to some prescription I ([5]). In the present context, an artificial molecule is an agglomerate of cells, which we propose to name "*cellicule*" (an artificial contraction between "cellular" and "molecule"). A few examples are shown in Fig. 3. Each cell is attributed state 0 or 1 (or white and black in the illustrations), thus defining a *state configuration pattern* on the cellicule. S denotes the symmetry (group) of the cellicule's *shape*. We will mainly consider cellicules with the cyclic group symmetries Z_1, Z_2, Z_3, Z_4, as illustrated in Fig. 3.

$\mathcal{C}$ denotes the set of all cellicules of given size (i.e., number of cells in the cellicule) and shape, but with distinct configuration patterns. For a cellicule of size n, the cardinality of $\mathcal{C}$ thus equals 2^n. P will denote a population of N cellicules of the same shape and size. N is the population size. Formally, it is a multiset

$$P = \{C_j\}_{j=1,\dots,N} = \{c_1(n_1), c_2(n_2), \dots, c_L(n_L)\},$$

where $C_j \in \mathcal{C}$ and $c_i \in \mathcal{C}$, $i = 1, \dots, L$, and n_i is the multiplicity of c_i in P. Obviously, $\sum_{i=1}^{L} n_i = N$.

We now present the so-called S-algorithm, which defines the dynamics D and the interaction I in a population of N cellicules with shape symmetry S.

The S-algorithm

step 1 : Initialize a population of N cellicules. This may be done by attributing randomly and independently state 0 or 1 to each cell in each cellicule.

step 2 : Implementing the dynamics D of the cellicules

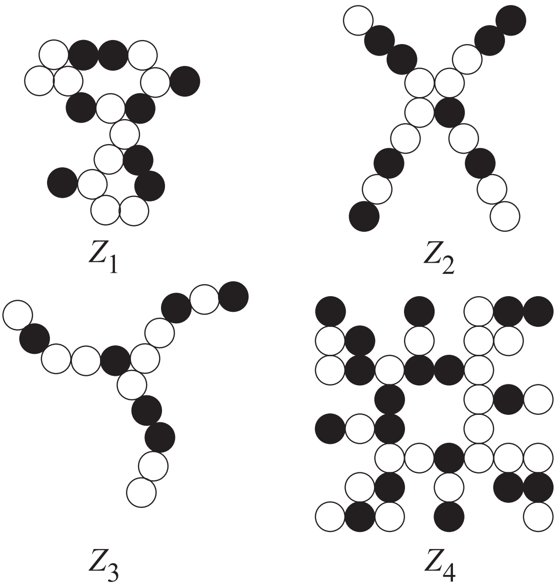


Fig. 3. Examples of cellicules with Z_1-, Z_2-, Z_3-,Z_4-symmetric shape. I.e., the shape remains invariant under the identity operation Id (for Z_1); under Id and Rev (=reflection along the vertical for the Z_2-case shown); under Id, and rotations of ±120° (for Z_3); under Id and rotations of 90°,180° and 270° (for Z_4). The state configuration patterns are not symmetric (or: have only Z_1-symmetry).

- substep 2.1 : Select randomly and independently two cellicules from the population, say C_a and C_b ($a, b \in \{1, \ldots, N\}$, $a \neq b$).
- substep 2.2 : Select randomly and independently two operations $\omega_1, \omega_2 \in S$, each with probability $|S|^{-1}$, and let them act on C_a and C_b respectively, producing the pair of cellicules

$$(\omega_1(C_a), \omega_2(C_b)).$$

- substep 2.3 : Align $\omega_1(C_a)$ and $\omega_2(C_b)$ by bringing all cells in these two cellicules in 1-1 correspondence through an exact juxtaposition of both.

step 3 : Implementing the interaction I between aligned cellicules

- substep 3.1 : For each cell in $\omega_1(C_a)$, find the corresponding aligned cell in $\omega_2(C_b)$, and keep or change the states in these cells according to the following rule:
  - $*$ if both aligned cells are in the same state: keep the cells in the same state
  - $*$ if both aligned cells have a different state: bring both cells into state 0 with probability ν_0, or to state 1 with complementary probability $\nu_1 = 1 - \nu_0$.

  This random interaction creates a pair of identical cellicules, each of which is denoted

$$(\omega_1(C_a), \omega_2(C_b)).$$

- substep 3.2 : Update the population by replacing both parent cellicules C_a and C_b by $R(\omega_1(C_a), \omega_2(C_b))$.

step 4 : Return to step 2.

Step 2 of this algorithm models the dynamics of a well stirred chemical reactor (in a sequential way): all cellicules are allowed to meet randomly in pairs and align properly, after which they interact randomly as described in step 3 (hence the qualification *random* artificial chemistry). The interaction can be seen as a random copying between two cellicules, whereby a cell in state 1 copies the state 0 of its aligned neighbour with probability ν_0 (equivalently: a cell in state 0 copies the 1 of its neighbour with probability ν_1). Of course, if aligned cells have the same state, they are already each other's copy, and nothing changes. This artificial chemical reaction can be symbolically represented by

$$C_a + C_b = 2R(\omega_1(C_a), \omega_2(C_b)).$$

It is clear that a population in which all cellicules have an identical S-symmetric state configuration pattern is a fixed point under the S-algorithm. But, due to the random dynamics and interactions, it is not clear that these fixed points are attractors. However, numerous experiments have led to the following

Observation 2.1 *The S-algorithm eventually brings the population in an equilibrium state in which all cellicules have the same S-symmetric state configuration pattern. And this no matter the population size, or the size of the cellicules.*

3 Analysis of the S-algorithm for $S = Z_1, Z_2, Z_3$

The observation can be turned into a theorem for $S_Z = Z_1, Z_2, Z_3$, formulated as follows

Theorem 3.1. *Let P be a population of N cellicules of size n, with S_Z-symmetric shape and with arbitrary initial state configuration pattern. The S_Z-algorithm will bring all cellicules into an identical S_Z-symmetric configuration pattern, with a probability tending to 1 as the number of iterations increases, whatever the values of N and n.*

The proof is based on showing that the *population state*, defined by the state vector $z = (n_1, n_2, \ldots, n_{L-1})$ performs a random walk in the corresponding population state space $\mathcal{S}_P = \{(n_1, n_2, \ldots, n_{L-1})|\ 0 \leq \sum_{i=1}^{L-1} n_i \leq N\}$, and that this random walk (modelled as a Markov chain) has absorbing states which correspond to populations in which all cellicules have the same S_Z-symmetric configuration pattern. It is hereby sufficient to consider populations of so-called elementary S_Z-cellicules, i.e., cellicules with S_Z-symmetric shape, but with a minimal number of cells.

For example, for studying the dynamics of the Z_1-algorithm, it is sufficient to consider a population of N single-cell cellicules. The population state is in this case $z = z_0$, i.e. the number of single-cell cellicules in the population which have state 0. A careful analysis of all transition probabilities induced by the different steps in the Z_1-algorithm shows that the underlying Markov model corresponds to the gambler's ruin random walk [6]. This translates in the present context into the fact that the population gets ultimately absorbed into state $z = 0$ or $z = N$, which corresponds respectively to all cellicules being in state 1 or 0.

When considering a population of N multicell cellicules under the Z_1-algorithm, it is clear that when looking in isolation at all cells located at a same (single) position in these cellicules, these cells evolve as a population of N single-cell cellicules under the Z_1- algorithm, and thus these cells will all end in either configuration state 0 or 1. As this will be the case for the cells in any particular position in the cellicules, it follows that all cellicules will eventually get into the same state configuration pattern (which will generally have Z_1-symmetry, meaning that the pattern is only invariant under the identity operation). When looking at the global picture: what happens is that there is a certain (random) position in the cellicules where all cells get into the same state first. Then there are no more changes at this position. Then there will be a second position in which all cells get into the same state, and so on, until finally all cells at all positions will have reached the same state (which may vary with the position).

A similar analysis is possible for the Z_2- and Z_3-cases, where population dynamics of elementary cellicules with respectively two and three cells have to be analyzed. This is still practically feasible, but it no longer is for other symmetries because of the high dimensionality of the population state space. See a forthcoming paper [4] for more details concerning absorption probabilities, and for generalizations to other symmetries, variations on the S-algorithm, and

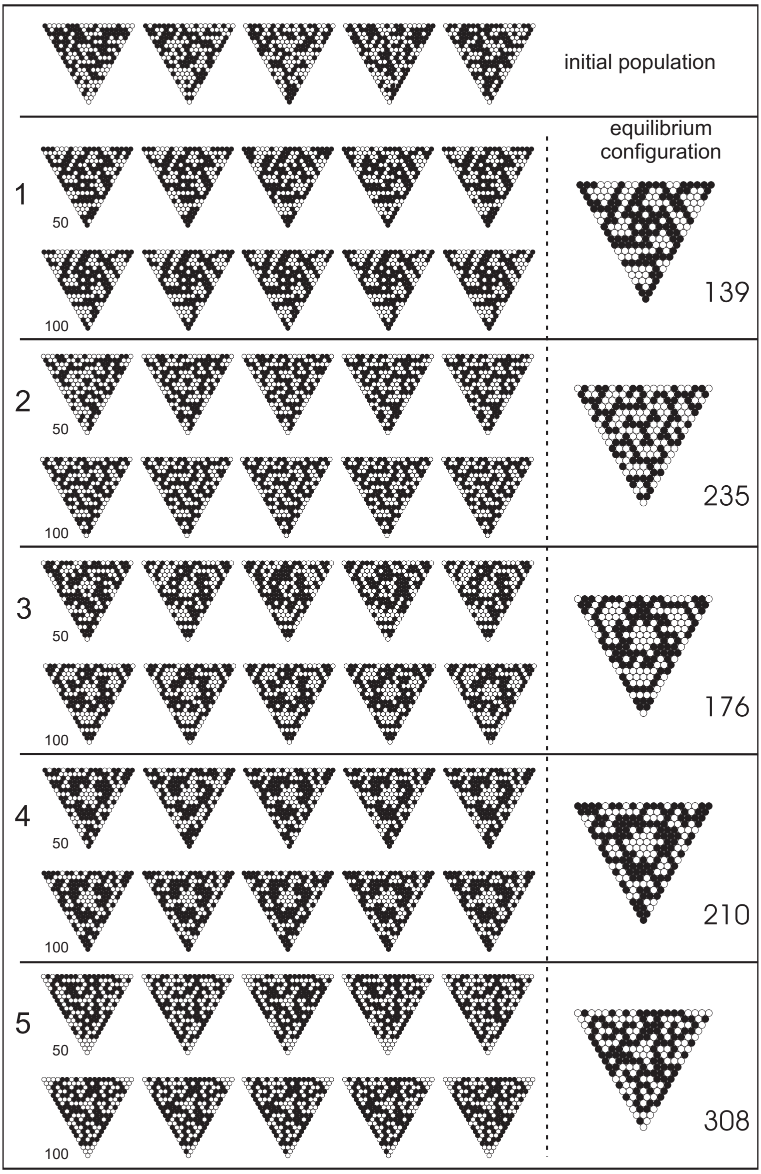


Fig. 4. 1-5: different runs of the Z_3-algorithm on a population of 5 triangular cellicules (having dihedral D_3-symmetric shape actually, Z_3 being a subgroup of D_3). The cellicules have 210 bi-state cells (20 cells along each side). These runs start with the same random initial population shown on top. Intermediate results for the population are shown after 50 and 100 iterations, as indicated. The last column shows the Z_3-symmetric configuration that all cellicules reach after the population has been absorbed in an equilibrium state, and is followed by the number of iterations until equilibrium for that particular run.

multistate cellicules. Figure 4 shows a few results for a population of 5 cellicules with Z_3-symmetric shape.

The artificial chemistry experiment that was presented above and that found its origin in the random feedback scheme reported in [1], as described in the introduction, clearly shows how randomness (which is present in the initialization of a population, in the selection and reorientation dynamics of the cellicules, and in the interactions between cellicules) may lead to the inevitable fate of uniformity in symmetry. This happens in a self-organizing way, without having explicitly being designed for this to happen (in contrast to the original feedback design which was an explicit purposeful construction). The implicit driving force is the S-symmetry of the cellicules' shape which constrains the random reorientations.

References

[1] Barbé, A. & von Haeseler, F. [2003] Symmetric self-organization in a cellular automaton requiring an essentially random feedback. Observations, conjectures, questions, Int. J. Bifurcation and Chaos **13**, 1029-1054.

[2] Barbé, A. & von Haeseler, F. [2004] The Pascal matroid as a home for generating sets of cellular automata configurations defined by quasigroups, Theor. Comp. Sc.**325**, 171-214.

[3] Barbé, A. & von Haeseler, F. Frame cellular automata: Configurations, generating sets and rlated matroids, article in press, Discrete Mathematics (2008), doi:10.1016/j.disc.2008.01.046.

[4] Barbé, A. [2008] Necessity from chance: self-organized replication of symmetric patterns through symmetric random interactions, SCD-SISTA Report no. **08-53**, Dept. of Electrical Engineering, KU Leuven, Belgium. To appear in Int. J. Bifurcation and Chaos.

[5] Dittrich, P., Ziegler, J. & Banzhaf, W. [2001] Artificial chemistries — A review, Artificial Life **7**, 225-275.

[6] Parzen, E. [1967] Stochastic Processes (Holden-Day, San-Francisco) Example 6A.

.

Signal crossing solutions in von Neumann self-replicating cellular automata

William R. Buckley

California Evolution Institute
San Francisco CA 94134
wrb@calevinst.com

Abstract. Signal crossing is a significant problem within von Neumann cellular automata (29-states); only three solutions are known. This paper gives a thorough examination of these three solutions, including discussion of performance and the effective means of construction, including the mechanism of auto-initialisation. Application of these solutions within self-replicators is then examined for Nobili and von Neumann architectures. Two self-replicators for Nobili cellular automata are presented. We demonstrate the transformation of cellular automata between these architectures; both configurations are transformed into von Neumann equivalents. We derive the signal crossing configuration having minimal footprint. We also present conjectures regarding signal crossing organs, suggest the nature of tapeless von Neumann self-replicators, discuss the distribution of complexity in self-replicators, and distinguish holistic self-replication from self-replication by means of partial construction, a novel mechanism.

1 Introduction

The mechanism of effecting control within configurations of von Neumann [17] cellular automata (vNCA) is translocated signal, best considered as a *packet* of bits bounded by values of one; the shortest signal is a single bit, or *pulse*. Generally, for two-dimensional cellular automata, a problem with translocation arises, known in vNCA as the signal crossing problem. An interesting characteristic of vNCA is that no state is defined specific to the service of signal crossing, service necessary to that beyond simple function. Instead, signal crossing mechanisms must be obtained by the aggregation of other states types. Further, all mechanisms must be constructible, for human abilities are expected to say nothing of the abilities of a machine.

Evidently, John von Neumann devoted a good deal of thought to the problem, discussing it with H. H. Goldstine [5] and Arthur W. Burks [3, 17]. Von Neumann was familiar with only one solution, a passive configuration known as the coded channel, *cc*, it being judged less than satisfactory even as it satisfied an ancillary requirement of his model of machine self-replication. By this we mean to imply an expectation of period researchers for the need of perfect signal crossing to the practical implementation of a vNCA self-replicator, a conclusion which Burks [4] has acknowledged respecting the von Neumann Memory Control.

Much effort was expended to improve the efficiency of this solution, with J. W. Thatcher [16], C. Y. Lee [7, 8], and S. T. Hedetniemi [6] each publishing various successive improvements upon von Neumann's design, the latest appearing in 1968; none produced a perfect signal crosser, one which crosses all signal without change. Clearly, the area of a self-replicator is proportional to the area of incorporated signal crossers and, given the state of automatic computing equipment in the 1950s and 60s, it was important for any demonstration of self-replication within vNCA that solutions of minimal area and maximal performance be found. Indeed, it is only within the current decade that personal computing equipment has produced sufficient throughput to demonstrate a vNCA self-replicator on the human time scale.

Hedetniemi also published another solution, a configuration that crosses two synchronous signals of arbitrary length, attributed to J. E. Gorman [6], and known as the real-time crossing organ, *rtco*. Though the *rtco* offers the perfect service unavailable in the *cc*, it is much more difficult to construct. Specifically, the *rtco* includes clocks, and the construction of clock bearing configurations presents its own difficulties. Burks [3, pp. 30] suggested one inelegantly effective means to construct the *rtco*, preserving von Neumann's requirement that a self-replicator be initially passive; started into action by an externally supplied pulse. Burks later brought the problem to the attention of Amar Mukhopadhyay [12], who in 1968 published his *mco*, a configuration functionally equivalent to the *rtco* but of a very different design. The *mco* is another clock bearing configuration. No prior work has been published respecting perfect signal crossing by means of a clockless configuration; our exposition touches on this issue.

Von Neumann designed his cellular automata being mindful of the need to minimise complexity while supporting non-trivial self-replication. Thus, while signal crossing is not a problem in three dimensions, von Neumann insisted upon a two dimensional model; it seems this is due the advice of Goldstine and Julian Bigelow [5, pp. 278]. Though signal crossing can be effected by the inclusion of special states into the model, von Neumann declined to include such states, also. Burks [17, pp. 261] reports no recollection of a reason for von Neumann's decision, suggesting state count minimisation (29 are specified) as the likely concern, though von Neumann [17, pp. 191] wrote of the issue. We offer that Occam would approve.

Breaking with von Neumann, Renato Nobili [13] added memory and signal crossing states to the model in the 1990s; ideal signal crossing became trivial. These changes increase the base complexity of the model, and reduce the functional complexity of configurations, but have little effect upon the overall complexity of self-replicators a la von Neumann. Configurations employing these extensions have corresponding near equivalent configurations devoid of these extensions. While signal crossing in Nobili cellular automata is easier than in vNCA, we will see that such extensions are superfluous; their function is easily obtained otherwise. Indeed, the primary effect of Nobili's extensions is upon configuration size and computational throughput.

We examine the three signal crossers, giving special attention to constructibility, and their use in self-replication, observing interesting results. We also present improvement to the Hedetniemi [6] design, deriving the likely minimal signal crossing organ for vNCA. First, we review some fundamentals.

Classification schemes for configurations shall be important to our exposition. The simplest partition is implied by a feature of construction; that all constructed states are passive, in that they carry no signal. This is, of course, a consequence of the vNCA state transition function. So, configurations are either passive or active, according to their carriage of signal. The labels *perfect* and *ideal* are used for signal crossers that (i) cross all signal, and (ii) do so within a configuration size of one cell, respectively.

Thatcher [16] defines cellular automata in geometric terms, relating cells to the cross product of the integers, and considers the configuration s to be the state of the entire space of cells for some time t. Passivity is then defined as the equality of a configuration with its transform under state transition. Thatcher also posits the notion of partial passivity, defining s as completely passive if every sub-configuration $s\acute{}$ is passive. We might imagine a partially passive configuration as one in which a long series of transmission states generally carry no signal, but for a few cells of the path. Hence, most of the configuration is passive, even though some small sub-configurations are not passive. The remainder of this discussion shall determine passivity upon the presence of signal, and define a configuration as any well specified grouping of cells.

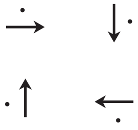


Fig. 1. An active sub-configuration $f(s') = s'$ which is identical to its state transition, and classifiable as passive or active, depending upon applied metric. No other figure shows signal, as this figure does; the dots adjacent to the ordinary transmission elements represent bits of value one; lack of a dot represents bits of value zero.

The organ shown in Fig. 1 is highly unlikely, as it would seem to serve no useful purpose. The more important consideration of passivity is upon the occurrence of purposeful active components, or clocks. Clocks are signal bearing and emitting, closed path configurations, and are unlimited in the duration of their behavior; the configuration in Fig. 1 is not a clock, as it is not a signal emitter. Though we directly address constructibility below, it is important to now take note that the occurrence of clocks within a configuration imposes limitations upon construction. A configuration becomes non-constructible when no means exist to isolate signal generated by active components from signal necessary to construction. That is, clocks external to the constructor supply unwanted signal to space under construction, thereby altering the constructed configuration. All completely passive configurations are constructible; each cell can be constructed without consideration of the state of surrounding cells. We shall see that means exist to avoid conflict between clocks and constructors; generally, unconstructible configurations have larger, constructible analogs. Figure 1 is the only figure of this paper that shows carried signal; the dots next to the ordinary transmission states.

The remainder of this paper is highly technical, a fact for which we offer no apology. Readers are well served by strong familiarity with von Neumann cellular automata; familiarity with Nobili cellular automata is highly recommended. In particular, command of the material and jargon contained within Buckley and Mukherjee [1], Burks [3], Mukhopadhyay [12], and Thatcher [16] is expected. We should mention also that many of the organs shown in the figures are in the form suggested by Thatcher [16], however optimised, these providing a standard by which to measure configuration efficiency and complexity, and reducing the difficulty readers may find in coming to a detailed understanding of the mechanisms of organ function. Readers are encouraged to obtain the computational resources mentioned at paper end, and use same to verify the content here presented.

Before moving to discussion of the mechanisms of construction, we should like to consider the question of what ought constitute complete specification of a self-replicator. By von Neumann's definition, the act of self-replication requires both constructor and description, these hence being suggested as necessary to any complete specification of a self-replicator. We assert this expectation, that proper specification of a self-replicator requires definition of both configuration and taped description, is without logical basis. Clearly, the description is totally dependent upon the configuration; it must be accepted by the configuration, and it must yield replication of the complex. Moreover, it is not possible to construct a tape description without detailed knowledge of the configuration to be described. Yet, from the mathematical perspective, an infinite set of tapes exists, each member having a fixed length; the size of the set is the sum of all the positive powers of two. This set is then partitioned in two by any given configuration; those tapes which result in self-replication of the configuration, and those tapes which do not.

For any self-replicable constructor, there is a set of tapes by which self-replication may occur, and that set of tapes is completely determined by the constructor. Hence, it is not necessary to specify a tape, in order to claim self-replicability. vNCA self-replicators are defined by their configuration, and the environment in which they exist. We assert that the superfluousness of the tape to complete vNSRCA specification is trivially obvious.

Rather more important is the behavior of the configuration, that the instruction set provides construction capability sufficient to self-replication. With but one exception - the time to halt. One generally expects in this case that tape length is as critical as is tape content. We deem a self-replicator to be its configuration, and determine size of a self-replicator solely upon the count of endo-perimetric configuration cells, exclusive of the tape. Thus, the minimal self-replicating configuration will not likely accept the shortest possible tape, nor consume the shortest possible time to self-replicate. We hold that configuration size, exclusive of tape, and upon which tape length and time to self-replicate are dependent, is the most important consideration of the three.

2 The mechanisms of construction

Construction is the process by which one configuration generates another. Non-construction in vNCA occurs in several forms. Some configurations cannot be constructed by any means, the minority of these being Garden-of-Eden configurations, GoE. Some such configurations have plausible pre-images, but no path to construction by act of construction arm. Other configurations impose functionality requirements upon constructors, and so are constructible by only a proper subset of all possible constructors. Thatcher [16, pp. 145] gives the first published example of a non-constructible configuration for vNCA, basing the argument upon the improbability of combinations of states, while as editor, Burks [16, pp. 138] adds a few comments in the paper footnotes. McMullin [10] writes that self-replicators can be augmented with any other machine, save those which "in some sense interfere with or disrupt the normal operation of" the self-replicator. Buckley and Mukherjee [1] present a second example configuration that is non-constructible, noting the local effects of constructor versus construct interaction.

An important issue respecting Thatcher's *sup()* operator is that it excludes those ground states which are contained within the perimeter of a configuration. Such endo-perimetric ground state cells are not necessarily non-functional vis-à-vis the configuration. Many undoubtedly insulate transmission states from confluent states, and are therefore an essential part of the configuration. Even where the organs of a configuration are optimally packed, internal ground states are to be found, and are consequential if not essential. We do not base configuration size measures upon *sup()*.

The reader should be disavowed of the construction of transition states. This does not occur. Rather, transition states are exactly that, transitions between the ground state and the passive states. Ascribing the non-constructibility of a configuration to the presence of transition states, as did Thatcher [16, pp. 145] by presenting a GoE, begs the question. The transitory nature of transition states makes clear that such states are not constructed. Non-constructibility more reasonably rests upon the nature of configurations of the non-transition states, and in particular, the nature of the constructor. One may speculate as to the potential for valuable construction obtained through the clever interaction of multiple signal sources and constructors. Perhaps the close interaction, spatial and temporal, of construction processes and signal sources are integral with the processes of development and homeostasis.

A constructor is a mechanism that causes cells to accept change, which for vNCA is an alteration of state class, as from confluent to ground. Hence, constructors are transmission states pointed at a cell that is either in the ground state, a transition state, or an annihilatable state. For special transmission states, annihilatable states are ordinary transmission states and confluent states. The construction arm is an articulate pairing of ordinary and special signal paths, these paths having the property of always being adjacent to each other. It is always the case that one of the signal paths terminates by pointing to the other signal path; the end of the construction arm. Thus, the construction arm sports two constructors. The constructor assumes that space under construction is in the ground state, just as construction arm articulation is assumed to be unimpeded.

XR	extend CArm right	25	S:.0000000000000101111110100. O:.1000110011000000000000001.
XU	extend CArm up	28	S:.1000000000000010011001011111. O:.0010001110011010000000000000.
RL	retract CArm left	30	S:.111011000000000000000011110100. O:.000000010100011100001000000001.
RD	retract CArm down	36	S:.100000010000000000000001101111110100. O:.001000110100111000110000000000000001.
OR	construct right ordinary transmission	5	S:.00000. O:.10000.
OD	construct down ordinary transmission	4	S:.0000. O:.1010.
OL	construct left ordinary transmission	4	S:.0000. O:.1011.
OU	construct up ordinary transmission	5	S:.00000. O:.10001.
SR	construct right special transmission	4	S:.0000. O:.1100.
SD	construct down special transmission	4	S:.0000. O:.1110.
SL	construct left special transmission	4	S:.0000. O:.1001.
SU	construct up special transmission	4	S:.0000. O:.1101.
CN	construct confluent	4	S:.0000. O:.1111.
IT	initialise signal transfer	5	S:.00000. O:.10001.
FT	finish signal transfer	13	S:.1101110000000. O:.0000000110000.

Fig. 2. A set of construction arm operations, the first thirteen of which provide articulation over the first quadrant, satisfy construction of all passive configurations, and facilitate transfer of signal between configurations. Special requirements of real-time crossing organ construction are satisfied by the additional two operations listed. Each operation is shown with its mnemonic, description, bit length, and signal. Sequences of operations can be implemented by the concatenation of signal. It should be noted that no provision is made for destruction of cell states, beyond that associated with construction arm articulation. Occasional necessary delay between signal is not shown.

The construction arm responds to signal alternately transmitted down each signal path. Clearly, construction arm integrity is a function of this signal; misplacement of a single bit within signal can result in the permanent disablement of the construction arm. The reader is invited to verify that such result is trivially achieved. The sequence of operations for construction are to cause the transition of state of the target cell, by delivering construction signal (including any necessary annihilation signal), followed by the retraction of the construction arm by one cell. The choice of path used for the delivery of signal is arbitrary save for one purpose; the transmission of signal from constructor to construct, which must come via the ordinary signal path. For example, the start signal used to initiate construct behavior. We should perhaps say, construct by any path; instruct by ordinary, only. Our examples construct and instruct using the ordinary signal path of the construction arm. Hence, they cannot destruct any construct, a fact having obvious implications regarding notions of universality over construction; these implications also apply to the joint and several work of Nobili and Pesavento [14, 15].

There are but two sources of signal; pulsers, and programmed controls. The pulser generates a specific finite signal for each pulse of input. Thatcher [16, pp. 147-148] gives the general pulser, a passive configuration that employs six cells for every generated bit of signal, delivered to one output channel. One can obtain two channels of signal with the dual-path pulser, by extending the general pulser to five cells wide, giving two generated bits (one each path) per ten cells. Such a configuration can directly drive the construction arm. A suitably large pulser can construct a self-replicator; we shall demonstrate this mechanism for construction of the *rtco*. We conjecture that a self-replicator is expressible as a dual-path pulser, albeit one having a more complex control structure than the in-line mechanism employed by Thatcher and von Neumann; the mechanisms of microprograms and hierarchically layered clocks come immediately to mind. This self-replicator has no tape, nor does it constitute a universal constructor. Indeed, the description is implied by, and expressed as, the configuration, it being instead a complex construct specific constructor, or *csc*; the description and configuration are an identity. Such a configuration is self-describing, and being initially passive, it is restartable.

A strong benefit of pulser use is the trivial assurance of proper timing of generated signal. This is especially important for signal that is to be transmitted from constructor to construct, such as that intended for component clocks. It is not so easy to obtain comparably accurate timing of signal generated by programmed controls. A programmed control is a high order configuration that generates signal according to an external, coded description of some configuration to be constructed, by the ordered triggering of a set of pulsers, each producing fundamental, purposeful signal. The process of converting description to signal provides many opportunities for corruption of signal timing. For instance, enforcement of time delays between signal pairs requires that a programmed control know such measures as the distance between description reader and interpreter, the distance between construction signal generator and construct, and the delay between interpretation and signal generation. The accuracy of such measures, and means to correspondingly adjust the timing of signal generation, is especially critical to programmed controls when the signals served by multiple clocks are to be configured with specific phase relationships.

The coded description of a configuration can be expressed as a sequence of operations, which when viewed in mnemonic form, has vaguely the appearance of assembly language; a construction program. Discussion of configuration construction may be qualified in terms of this form. Further, knowledge of signal associated with each operation makes possible quantified comparison between various construction arm behaviors. We use this process in our analysis of *rtco* and *mco* construction. A total of thirteen operations suffice to map the first quadrant of space, to construct all passive configurations, and to pass signal between configurations. These and other operations are listed in Fig. 2. The signal necessary to construction is produced by concatenating the signal for each operation of the program, in the required order. The additional operations listed in the figure are necessary to construction of the *rtco*, and other clock bearing organs. Nota bene — these operations are insufficient to universal construction.

The last mechanism of construction we consider is auto-initialisation, a means of organ start first suggested by Mukhopadhyay [12]. An organ is auto-initialised by the first input signal, which is sampled, processed, and distributed as required by an auto-initialisation circuit to the component clocks of the configuration. Auto-initialisation can be extended, so that an organ is initialised for every input. This allows for organ function reassignment through organ reconstruction. Lee [7, 8] uses a primitive form of this concept in his design of a cellular computer, modifying the memory pulser of the memory assembly between two configurations, according to the state of the bit stored. We shall see that auto-initialisation offers far more efficient means to construct active organs than does a dedicated pulser, minimising the complexity of programmed controls by moving construction complexity to the tape encoded description, and to the construct. Auto-initialisation is demonstrated for *rtco* construction, and we compare the efficiency of this method to that offered by task specific pulsers, a *csc* for example, and programmed controls.

3 The coded channel

The *cc* is the only signal crossing organ designed by von Neumann. It is a non-cyclic, controlled access signal path of finite length, capable of serving an arbitrary number of arbitrarily large fixed length non-interfering asynchronous signals. The signal path is accessible via two layers of ports; the input layer allows the passage of signal into the *cc* signal path, and the output layer allows the passage of signal from the *cc* signal path. In the von Neumann design, an example being shown in Fig. 3, a port is a pairing of decoder and pulser. The decoders detect the presence of signal of some form and length, outputting a pulse to indicate detection; the pulsers emit signal upon receipt of a pulse. The pairing of decoder and pulser amount to a crude translation mechanism, detected signal being converted into a form suited to transmission via the *cc* signal path (the code of the channel), this signal then being converted a second time, to suitable output form. It should be noted that the decoder is able to distinguish only those one valued bits of signal; zero valued bits are ignored. Hence, the decoder is an ambiguous signal acceptor, best when cleverly used. One may improve signal selectivity on input by replacing the decoder with the recogniser, at the cost of increased port size. An important note of caution is that signal longer than that accepted by a recogniser or decoder is ambiguous when visited upon those organs. The size of the *cc* is proportional with port size and count.

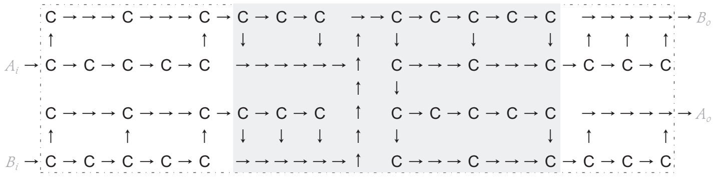


Fig. 3. The general coded channel. This organ crosses two signals, `101` and `111`, each detected on a different port, and the same signals represent themselves within the signal path. This organ covers a rectangular region of 33×7, or 231 cells, of which 83 are in the ground state, or ∼ 1/3. The shaded portion is comparable to the example given in von Neumann [17, pp. 336].

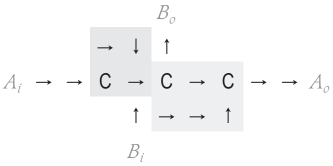


Fig. 4. The degenerate `101` coded channel of the von Neumann triple-return counter. The signal at input and output is a pulse. Only one path is protected by decoder and pulser, so signal from input A_i is sent to output A_o and B_o, while signal from input B_i (being a single bit) is sent only to output B_o. The pulser and decoder shown are of minimum size and area, and are shaded for easy identification.

The *cc* is an extremely flexible organ, and ports of either type may be placed at any point along its length, allowing signal to be selectively gathered any number of times, and similarly deliverable to many destinations. Corruption may occur upon the admission of unsuitably synchronised signal; less than 100% of incident signal is crossed without corruption, herein known as *unserviced signal.* Though some corruption is correctable, it is not generally so, and such limitations become serious concerns in higher order configurations having the *cc* as a component. The *cc* is not a perfect signal crosser.

Another measure important to signal crossing organ performance is the rate at which signal is crossed, termed frequency. That is, frequency is the inverse of the time delay required between incident signal such that crossing occurs. Frequency applies independently to each signal path. An organ which provides the ability to cross all signal, does so at the frequency of one along all paths. An organ which provides the ability to cross at the rate of one signal every eight clock ticks serves at the frequency of 1/8. For notational simplicity, frequency is often expressed as the time delay itself, instead of its inverse. It must be remembered that this delay is different from that of signal propagation through the organ, which is the meaning used in historic literature.

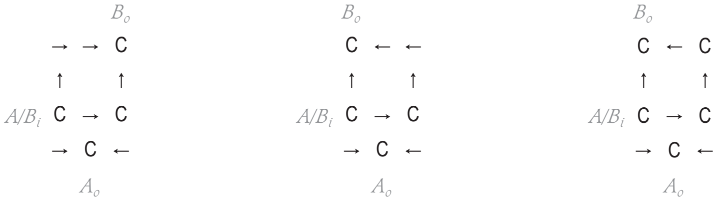


Fig. 5. The `1001.11`, `1001.100001`, and `1001.10000001` signal switch crossing organs.

Various degenerate forms of the *cc* have been explored, von Neumann being here also the first, such as is used within an organ known as the triple-return counter, *trc*, a device for computing long delay. Burks [17, pp. 180] writes of a possible *trc* malfunction, noting the possibility "is a special case of the general problem of" signal crossing. Von Neumann [17, pp. 183; both quotes] discussed *trc* design, noting that while the interconnections between some organs were easily made, "serious topological difficulty" exists in making other interconnections, following with "It is clear that these paths must cross." Subsequently, von Neumann puts forward the general case and its solution, the *cc*. Von Neumann nevertheless resolved that for *trc* design, the better solution is obtained ad hoc. Undesirable output from the signal crossing configuration does not corrupt *trc* operation, so the configuration is sufficient for its application, assuming that the *trc* is not restarted at A_i during operation. We shall see that this form presages what we present as the minimum possible vNCA signal crossing organ. The relevant *trc* sub-configuration is shown in Fig. 4.

Some degenerate forms of the *cc* are obvious. For instance, if the signal input to a port is a single bit, then the corresponding decoder can be eliminated. If an input to the *cc* is already in the form of channel code, then there is no need for the input pulser, either. Further, if single bit output is the desired result of signal crossing, then one need only the output decoders. Hence, signal to be crossed can be directly introduced into the *cc* signal path; one need only the left part

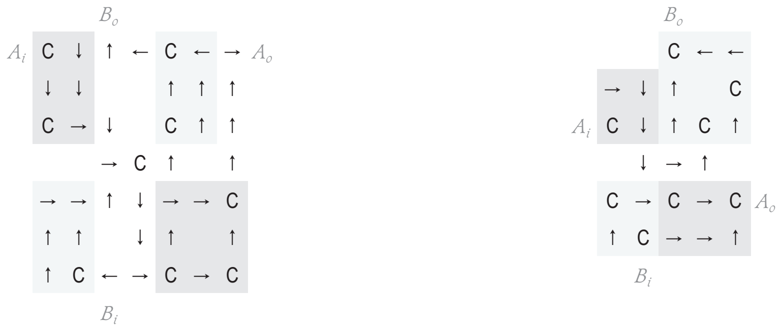


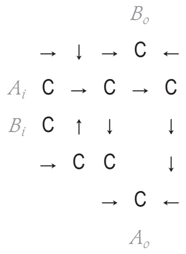
Fig. 6. The signal crossing organs of Lee, shown at left, and Hedetniemi, shown at right. The Lee organ uses the `11.101` channel code. The Hedetniemi organ uses the `101.1001` channel code. Critical components, decoders and pulsers, are shaded, the rest being component interconnect, with the Lee organ including formatting to keep the area square. These organs logically correspond to the inner half of Fig. 3. The Hedetniemi organ is nearly 39% smaller in size. While more space efficient than the Lee organ, it is less efficient at crossing signal, being more susceptible to signal interference.

		Lee	Hedetniemi
Propagation Delay		25 / 20	13 / 17
Unserviced Pairs	*A*	$\{1,\ 0^2 1,\ 0^3 1\}$	$\{1,\ 0^3 1,\ 0^5 1,\ 0^6 1,\ 0^7 1\}$
	B	$\{1,\ 01,\ 0^3 1,\ 0^4 1\}$	$\{1,\ 0^2 1,\ 0^3 1,\ 0^4 1\}$
Count / Range		6 / 8	8 / 12
Area / Size		7x7 / 41	5x6 / 25
Frequency		5 / 5 :: 7	6 / 7 :: 7

Fig. 7. The performance measures of the Lee and Hedetniemi signal crossing organs. Unserviced signal pairs are grouped by the varying input versus the signal `1`; the labels *A* and *B* indicate the input which does not vary. Range is the maximum time separation between two sets of uncrossed signal pairs. Size is the number of non-ground states in the area covered by the organ. Frequency is given for one channel operation to the left of colons. Two channel operation is to the right of colons, and assumes minimal interpath signal phase difference.

of the right half of the configuration of Fig. 3. Indeed, if one chooses either of the three signal pairs `1001.11`, `1001.100001`, or `1001.10000001`, then one can construct the signal switch crossing organ in a 3×4 region, using eleven cells, as shown in Fig. 5. Other variations of the signal switch shown can be constructed in a 3×5 space, again using eleven cells.

Size minimisation of the shaded portion of Fig. 3 has been the chief effort of Lee and Hedetniemi, who demonstrate sequential design improvements in respective subsequent papers. We now review these designs, first presenting their forms, then presenting analysis of their relative performance. This is followed by a short analysis of signal crossing fundamentals, leading to specification of the likely minimal footprint signal crossing organ for vNCA. The Lee and Hedetniemi organs are shown side-by-side in Fig. 6; performance characteristics for both are shown in Fig. 7. Comparison between these organs is best accomplished by considering those cases in which two closely incident signals are successfully crossed. Each case is found by varying the time separation between signal input. Clearly, given some suitably large time separation, all signal is crossed without interference. The problem to be solved is however not the crossing of signal for which interference is not a possibility.



Propagation Delay		14 / 13
Unserviced Pairs	*A*	$\{1,\ 0^2 1,\ 0^3 1,\ 0^4 1\}$
	B	$\{1,\ 0^3 1,\ 0^5 1,\ 0^6 1,\ 0^7 1,\ 0^8 1,\ 0^9 1,\ 0^{10} 1\}$
Count / Range		11 / 15
Area / Size		5x5 / 21
Frequency		6 / 3 :: 12

Fig. 8. The reduced Hedetniemi organ and performance measures. While smaller, the reduction in signal crossing capacity is the greater change.

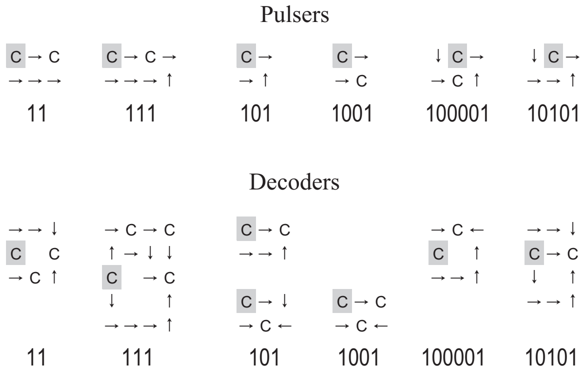
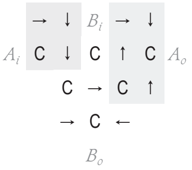


Fig. 9. Basic signal generators and detectors of minimal size. Input is to the shaded cell, and output is generally to the right for all organs, with the associated signal in caption below the organ.

Hedetniemi [6, pp. 4] claims his design is an "improvement of the 7×7 crossover network of C. Y. Lee." We can see from the performance measures shown in Fig. 7 that the claim is problematic. The Hedetniemi organ is approximately 39% smaller, at the cost of a 33% increase in the number of unserviced signal pairs; it is less able to serve signal crossing. However, the signal propagation delay is significantly reduced, by as much as 48%. Further, the frequency of two channel input is equivalent, for at least some phase relationships between incident signal. While the minimisation of signal propagation delay is important to overall configuration performance, such as that of a self-replicator, it happens that delay in vNCA configurations is generally a good thing. If signal crossing alone is the criterion, then perhaps Hedetniemi's claim is wrong. We can do better, producing further reduction in organ size, and some corresponding reduction in signal crossing performance.



Propagation Delay		14 / 9
Unserviced Pairs	*A*	{01, $0^4$1, $0^7$1, 0^{10}1}
	B	{01, $0^2$1, $0^3$1, $0^4$1, $0^5$1, $0^6$1}
Count / Range		10 / 17
Area / Size		4x5 / 16
Frequency		7 / 6 :: 12

Fig. 10. The minimal signal crosser for two pulses, the 4×5 *cc*. Several variations of this design occur, with areas of 4×7, 4×6, and 3×9, each having different performance measure. The shaded portions correspond to the organs of the degenerate *cc* found in the *trc*.

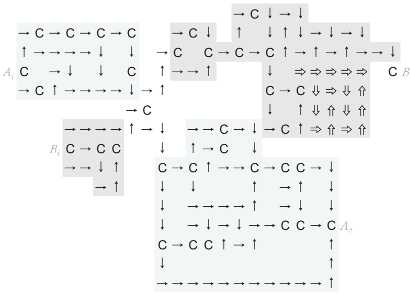
	Lee	Hedetniemi	Reduced Hedetniemi	4x5	Filter Corrected
Propagation Delay	25 / 20	-48 / -15	-44 / -35	-44/ -55	+200 / +260
Count / Range	6 / 8	+33 /+50	+83 /+88	+67/ +213	-33 / +75
Area / Size	49 / 41	-39 / -39	-49 / -49	-59/ -61	+765 / +441
Frequency	5 / 5 :: 7	-20 / -40 :: 0	-20 /+40 :: -71	-40/ -20 :: -71	-40 / +40 :: -86

Fig. 11. Relative performance between presented cc configurations. Values are given in percent of change from the Lee organ, rounded to the closest whole number. Improved performance has a positive sign. Frequency of two channel input (right of colons) is derived from pulse pairs of minimal phase difference; this would be signal pair `1.1` for the 4×5 *cc*. Other possible pairs having larger phase differences are `1.0`^{7}`1` and `0`^{2}`1.1`; such other pairs may yield different and perhaps better two channel input frequency for some organs.

The explicit separation of the upper and lower shaded portions of the Hedetniemi organ is easily eliminated, yielding a 5×5 form. This reduced Hedetniemi organ, shown in Fig. 8, has a 17% smaller area, 16% smaller size, and 38% reduced signal crossing performance. We see therefore a trend, that signal crossing performance varies inversely with reduction in organ size, and would like to determine the size of the smallest crossing organ, and its utility.

To derive the smallest possible *cc*, one need start with its components; decoders and pulsers. Figure 9 presents those decoders and pulsers of minimal size corresponding to a set of short signals. We see that the smallest pairings of decoder and pulser occur for the `101` and `1001` signals; these represent the pulsers and decoders of minimal possible size for vNCA. Ignoring potential undesirable interactions between adjacent organs, one should expect that a signal crossing organ could be constructed for this signal pair in a space no larger than 4×5. Presently, we show that this can indeed be accomplished.

One can easily avoid signal conflicts between these four organs within a configuration having an area of 4×6 and a size of 19, an organ that shares one cell between both decoders. Minimisation of configuration requires maximisation of configuration sharing. Sharing of configuration between functions requires that clocks not be introduced into the configuration, and generally requires clever arrangement of component organs. For example, one can construct a 3×3 configuration that includes both pulsers, while having a size of seven cells, not eight; the area increases even as the size decreases. As already suggested, one can share a confluent state between the decoders for these signals, saving one cell. The key trick is to share part of the pulser configuration with part of the decoder configuration, in this case resulting in a two cell reduction. We see these optimisations in the configuration shown in Fig. 10, together with the performance measures. The resulting configuration performs marginally better at signal crossing than does the reduced Hedetniemi design. Comparison of all these *cc* organs is presented in Fig. 11.



Propagation Delay		50 / 52	
Unserviced Pairs	*A*	$\{0^7 1,\ 0^8 1\}$	
	B	$\{0^7 1,\ 0^8 1\}$	
Count / Range		4 / 17	
Area / Size		25x15 / 181	
Frequency		7 / 3 :: 9	when synchronous
		17/18	when asynchronous, with phase distortion in channel *A*

Fig. 12. A filter-corrected *cc* that crosses all `1011101.1100011` channel code signal pairs, shown with performance measures. As shown, this organ is not compressed to its minimal supporting area.

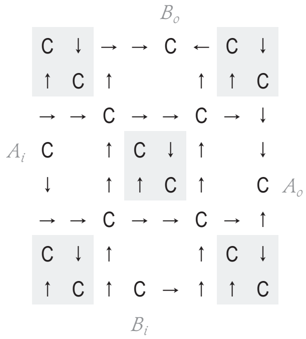
No *cc* crosses all signal pairs; at minimum, four signal pairs are not crossed. Yet, unserviced pair count is minimised through careful selection of channel code, as is the complexity of resultant corruption to organ output, allowing correction by filters on *cc* output signal, to yield crossing of all signal pairs, with occasional phase distortion, at a frequency substantially but not horribly less than one. This is the case for the `1011101.1100011` channel code; the *cc* is shown in Fig. 12. We conjecture that one may cobble together perhaps a great many such crossing organs, distributing the workload across a correspondingly large number of signal crossing paths, and so produce a perfect if monstrous signal crosser which is completely passive. The development effort is left as a reader exercise.

We have shown that the minimum *cc* for two pulses is a 4×5 configuration that uses the `101.1001` channel code signal pair, and that the minimum degenerate *cc* is the signal switch, a 3×4 configuration with three variations, each having a different channel code signal pair; these results have been neither previously demonstrated, nor published. We shall later show that the 4×5 *cc* is sufficient for construction of a vNCA self-replicator, which is itself an isomorph of an NCA self-replicator that employs only ideal signal crossers. Presently, we begin our analysis of the *rtco* by comparing its performance with that of the 4×5 *cc*.

4 The real-time crossing organ

It is a maxim of engineering that the fewer contingencies with which the designer must contend, the easier is implementation of problem solution. The failure of the *cc* to cross some signal pairs of channel code without the aid of filtering, and the argued necessity for monstrous *cc* size in order to service packet signal, suggests that crossing would be better served by a different class of organ. We have already seen that there exist but two classes of configuration within vNCA, passive and active; it would seem that we need an active signal crosser. Interestingly enough, the fundamental mechanism used by both the known active signal crossing organs, the *rtco* and *mco*, is to spread the workload across several signal paths, as suggested in the above reader exercise. The use of this technique is optimally applied in the *rtco*; no smaller perfect signal crosser exists for vNCA. Nota bene - vNCA signal crossing efficiency is limited to a rate of 50% in each direction of signal transmission. Perfect signal crossing therefore requires two signal paths in each direction of signal transmission.

The introduction to this paper mentions that the construction of clock bearing configuration is moderately to impossibly difficult, and that the construction of passive configuration is trivially easy. Other discussion mentions the knowledge and components necessary to a programmed control which is able to accurately time construction signal. To these we should add more exotic techniques, like the use of external configurations to augment construction arm function, pulsers dedicated to specific construction tasks, and auto-initialisation. All of these mechanisms are discussed in this section. We will see that the benefits of perfect signal crossing are quite costly to obtain by all of these mechanisms, save auto-initialisation; even then, perfect signal crossing is not free. Yet, the construction of self-replicators within vNCA is not particularly difficult. Further, no necessity exists for perfect signal crossing, even as packet signal is served. The *rtco* is shown in Fig. 13.



C	↓	→	→	C	←	C	↓
↑	C	↑			↑	↑	C
→	→	C	→	→	C	→	↓
C		↑	C	↓	↑		↓
↓		↑	↑	C	↑		C
→	→	C	→	→	C	→	↑
C	↓	↑			↑	C	↓
↑	C	↑	C	→	↑	↑	C

Fig. 13. The *rtco*, shown without clock signal. Each clock carries and synchronously emits $1\overline{01}$ signal.

Burks [3, pp. 16-18, 30] [17, pp. 262] examines the *rtco* in great detail, including the mechanisms of signal propagation within the organ. In summary, we note the *rtco* is an 8×8 composite of four 5×5 signal crossing elements, and four signal paths. The signal paths are grouped into parallel pairs, the pairs orthogonal to each other, giving two channels, *A* and *B*. All signal paths are of identical length; the channels begin and end with a cell in the confluent state. The first confluent cell duplicates signal, and the last selects which duplicate to output, from disjoint signal halves, these being transmitted via alternate channel paths. Signal crossing elements are composed of a pair of clocks emitting synchronous $1\overline{01}$ signal, the pair diagonally bounding a cell in the confluent state which has two input transmission cells, and two output transmission cells. Each element provides an alternating 50% service rate each to its two signal paths, this signal coming from one of the two clocks; no transmission cell gets signal from both clocks. Each path of the *rtco* is common to, and serviced by, two signal crossing elements. Hence, the *rtco* is a perfect signal crosser, though not an ideal signal crosser, like that produced by Nobili [14]. The bits of signal are partitioned into two disjoint sets, one each path, by the operation of the signal crossing elements. The central clock is common to the four signal crossing elements; the *rtco* could just as well be constructed from eight clocks, and be distributed over a very much greater area. This does not make construction easier, even if it makes clocks more accessible.

In a footnote, Burks [3, pp. 30] describes a multi-step process for *rtco* construction. The greater portion of the *rtco* is constructed, save a two-cell wide path giving the construction arm access to the central clock. Then, auxiliary constructs are started synchronously, producing signal to configure the clocks synchronously, and complete construction as needed. The footnote is:

> *"One can use a real-time crossing organ in an initially quiescent automaton in the following way. Construct the quiescent part of the crossing organ, except for a path to the center clock. Then add pulsers to complete*

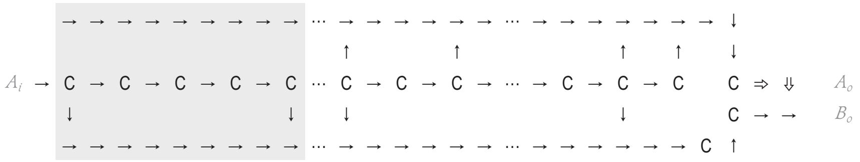
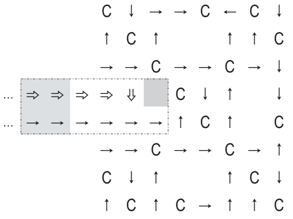


Fig. 14. The dual-path pulser, configured as a simple construct specific constructor. The construction arm of this organ is controlled by the signals shown in Fig. 12. The shaded area pulses construction of one upward pointing ordinary transmission element at the end of the construction arm's ordinary signal path; an area of 45 cells. This is in terms of space utilisation a very inefficient way to generate construction signal.

> *the construction and start the organ. The starting stimulus to the whole automaton must stimulate these pulsers, which then complete and start the crossing organ."*

To address Burks' method, we must first understand the nature of any such external augmentation of the construction arm. This analysis necessarily includes the operation of programmed controls, and the form and content of external descriptions; our starting place. One expects *rtco* signal paths to be intersected by other signal paths carrying start signal to the auxiliary pulsers but, there is no infinite regress here. Paths need only overlay, and additional pulsers can be added along the trigger signal path so as to complete construction of the *rtco* signal paths, and other configuration. Our analysis is only of the pulsers necessary to *rtco* construction, not those necessary to completion of other intersected signal paths; we discuss only the tip of the iceberg.



1. OD OR {*starts the clock*}
2. BR OU_2 RL_2 OD XU CN
3. RD RL

```
S:  .1101110000000.    .kill.SR.kill.0.    .  .
O:  .0000000110000.    .    .  .    .0.kill.OR.
```

Program signal length is 205 bits. Here, we see necessary delay between signal, for the BR instruction. This delay serves to avoid interference between construction arm constructors.

Fig. 15. The rtco under construction by Burks' method. The special path of the construction arm (outlined) must not terminate in the right-most shaded cell. Expected construction arm function requires a signal not listed in Fig. 12. The bridge is removed by the instruction **BR**; the signal is shown, with derivation. The group of four shaded cells is the location of the construction arm at the end of signal transmission.

Von Neumann understood the value of complexity distribution, his constructor off-loading the complexity of the construct (to the degree allowed by the environment) onto the external description. A constructor has a required minimum complexity, which is totally dependent upon an environment; this complexity being that which is necessary and sufficient to controlled manipulation of that environment. Any other complexity within the constructor is to the benefit of the construct, not the constructor. It is intuitively obvious that the simplest programmed control is the one which makes no decision of its own, following the external description blindly; there is no code compression scheme applied to the external description. While less intuitive, it is also obvious that the shortest external description is the one with the least complexity. We note that there are two kinds of complexity within the tape, associated with (a) construction arm activity, and (b) the complexity of the construct. Excess type (a) complexity is of no benefit to the constructor.

The problem with Burks' method is that it does not move complexity as completely as possible to the construct, and in consequence the construction arm performs a great deal of work that does not directly yield a functioning *rtco*. The construction of supporting pulsers amounts to the production of much detritus, as these pulsers have no purpose beyond *rtco* completion. Thus, they impose a significant size cost to any configuration employing an *rtco* constructed by Burks' method. To understand the scale of this size cost, we review large pulsers like the *csc*. The general *csc* is shown in Fig. 14; complex *csc* design is not addressed in this discussion.

A construction program can be encoded in two forms; the instruction on tape, and the bit pattern pulsed by a *csc*. Of critical concern is the size of pulser Burks would need for (i) starting the central clock, and (ii) completing those portions of the *rtco* obscured by the extended construction arm. Figure 15 shows Burks' configuration. We see that a right pointing ordinary transmission element is required to bridge the distance between construction arm and the *rtco* central clock, for by ending the special signal path of the construction arm in the position of the shaded cell, construction becomes impossible; signal transmitted to the clock would be re-transmitted to the construction arm. Hence, the construction arm needs signal sufficient to remove this bridge, signal which is not present in the instruction list shown in Fig. 12. This additional signal is shown in Fig. 15, and the operations necessary to complete *rtco* construction require a signal length of 205 bits. The *csc* covers an area of 2050 cells. This is an area 32 times the area of the *rtco*, and just over 340 times the area constructed; six cells are constructed.

In absorbing this point, that the area of the simple *csc* is many times the area of its construct, recall that four other pulsers are required to synchronise signal in the other four clocks of the *rtco*. Further, and recalling Burks purpose, to solve the signal crossing problems of the von Neumann Memory Control by employing four *rtco* configurations, the reader should understand that all twenty of these pulsers are supplied with the same start signal as that which starts the construct of which they are a part. This mechanism of construction is intolerably wasteful. Compression of the signal would require the development of highly complex clocking mechanisms, an effort expected to be much in excess of the effort to produce a general constructor; a self-replicator is very much simpler. Moreover, regardless of size or efficiency, the observation of such detritus within mathematical models is anything but elegant.

The problem gets much worse. What Burks did not in writing consider is the direct construction of the *rtco*, as might be mediated by a *csc*. The trivial assurance of clock synchronisation is for the *rtco* a highly desirable benefit. The immediately valuable question is, can that benefit be obtained for reasonable cost? We use the mnemonics shown in Fig. 2 to derive the message generated by this pulser. Construction follows the general sequence of (i) extending the construction arm by some length η, followed by a sequence of (ii) η pairs of one construction operation followed by one construction arm retraction. So, each cell constructed requires one extension, one retraction, and one construction operation. The sequence of construction arm actions necessary to *rtco* construction by *csc* is given in Fig. 16. We assume the construction arm is positioned with the ordinary signal path pointing at the lowest left cell of the region in which the *rtco* is to be built.

While the *rtco* is constructible via simple *csc*, yielding active *rtco* subconfiguration even as adjacent configuration is under construction, the corresponding *csc* is prohibitively large. From the *rtco* construction program, and the lengths of the corresponding signals, we determine the number of bits reasonably required to efficiently construct the *rtco*. Computation by this method yields the answer of approximately 6,500 bits per channel required for *rtco* construction. Such a value corresponds to a simple *csc* having a size of approximately 65,000 cells, which is several times larger than the size of a known vNCA self-replicator, not to mention being in excess of 1000 times the size of the *rtco*. The rotational asymmetry of construction signal within vNCA gives eight variations of the *rtco*; two pairs of mirror images about each of the co-ordinate axes, each with different construction programs. General solution of signal crossing by *rtco* requires construction of four, non-mirror variants, and therefore a proportionately larger *csc* area. Clearly, so simple a pulser as the simple *csc* is, on basis of space efficiency, a very poor choice for *rtco* construction, and indeed for the construction of any configuration, active or passive. A complex *csc* would likely be many times larger.

It is our expectation that design of a constructor capable of constructing the *rtco* by means of programmed control requires substantially greater effort than does design of a self-replicator. For instance, one cannot reliably synchronise construction signal via a programmed control without exact knowledge of the internal timing of the constructor. The acquisition and maintenance of such knowledge was discussed by Thatcher [16, pp. 169], who addressed a requirement of von Neumann's self-replicator design, that the position of the read/write head along the tape need be known in order to alter the message there stored. Thatcher argues the von Neumann design to be unnecessarily complex, that signal should not be dependent upon delivery distance, and demonstrates simpler means to alter tape content. What is common to both mechanisms is the

1.	XR$_3$	XU	XR$_4$								
2.	CN	RL	OU	RL$_3$							
3.	XU	XR$_4$	OD	RL	CN	RL$_3$					
4.	XU	XR$_4$	OU	RL	OR	RL$_3$					
5.	XU	XR$_4$	CN	RL$_4$							
6.	XU	XR$_4$	OD	RL$_4$							
7.	XU	XR$_4$	OD	RL$_4$							
8.	XU	XR$_4$	CN	RL	OU	RL$_3$					
9.	XU	XR$_4$	OD	RL	CN	RL	OL	RL	CN	RL	OR
10.	RD$_2$	XR$_3$									
11.	IT	——	OD	OR	FT						
12.	OR	RL	IT	OU	FT	CN	RL$_2$				
13.	RD$_2$	XR$_2$	IT	OU	RL$_2$						
14.	RD	XR$_2$	CN	RL$_2$							
15.	RD	XR$_2$	OU	RL$_2$							
16.	XU	XR	OR	RL							
17.	RD$_3$	XR$_3$									
18.	IT	$\varphi_1{}^b$	OD	OR	FT						
19.	RL$_3$										
20.	XU	XR$_2$	OU	RL	OR	RL	CN				
21.	XU$_3$	XR	CN	RL	OU						
22.	XU	XR	OD	RL	CN						
23.	XU	XR	OR	RL	OR						
24.	RD$_6$	RL$_2$	XU$_5$	XR	OU	RL					
25.	XU	XR	CN	RL	OR						
26.	RD$_3$	XR$_2$									
27.	IT	$\varphi_1{}^c$	OD	OR	FT						
28.	OR	RL	IT	OU	FT	CN	RL				
29.	RD$_3$	RL									
30.	XU$_7$	XR$_2$	OU	RL	CN	RL	OU				
31.	XU	XR$_2$	OR	RL	OD	RL	CN				
32.	RD$_2$										
33.	IT	$\varphi_1{}^d$	OD	OR	FT						
34.	OR	RD$_5$	XR$_2$	OU	RL	CN	RL	OU			
35.	XU	XR$_2$	OU	RL	OD	RL	CN				
36.	XU	XR	OR	RL	OR						
37.	XU	OD									
38.	XU	CN									
39.	RD$_5$										
40.	IT	$\varphi_1{}^e$	OD	OR	FT						
41.	RL										

Fig. 16. The mnemonic form of the construction program for the *rtco*. The program given is optimised for size, and totals 289 instructions plus a few hundred bits for necessary separation between signal; ~6500 bits. The program for Burks' scheme is considerably larger. The zero valued bits φ are used to correct the phase of signal transmitted to clocks, and have either a length of zero, or of one. The construction arm returns to its initial position upon servicing the bit sequence corresponding to these instructions.

distance between programmed control and the read/write head along the tape, upon which distance signal transmission time is dependent. As the control of construction signal timing involves this and other variables, a corresponding programmed control would be substantially more complex than is von Neumann's self-replicator design. Such capability increases the complexity of a constructor, instead of its construct, and consequently the effort of the designer. Note that we have not addressed the not insignificant increase in tape length necessary to support of Burks' method. We leave the effort to produce a programmed control capable of *rtco* construction as an open problem, turning our attention to auto-initialisation.

Auto-initialisation provides much more capability than that used in the *rtco*, the examples of clock synchronisation and trivial reconstruction being the simplest applications. Mechanisms used are signal sampling, portal closure, clock synchronisation, and staged initialisation. Our implementation uses three separate start signals; a staged initialisation. Each clock of the *rtco* has a separate auto-initialisation circuit. Signal to a stage is intercepted from a suitable path of channel A, by an interposing confluent state, or portal, and directed to the pulser of the adjacent auto-initialisation circuit by an auxiliary path. Pulser output is directed to clock input; and to a special transmission path constructor, which is addressed shortly. Clock signal for the entire organ is suitably synchronised, as initialisation of the first clock pair occurs simultaneously upon input of the first start pulse. All subsequent initialisations occur given knowledge of timing for the first two clocks. In this way, synchronisation becomes a trivial concern. Figure 17 shows the auto-initialisable *rtco*.

The minimum start signal for the configuration shown is $\mathtt{10^{13}10^{32}1}$. Signal with shorter lengths of zero bits result in a failed initialisation. An odd number of zero bits must precede the second pulse. For the third pulse, and even number of preceding zero bits is necessary. A pulse occurring at other intermediate times is ignored; one generally expects supply of multiple pulses to auto-initialisation circuits yields a failed initialisation. Unless supplied as a special signal generated by the constructor, the above signal is more easily generated by a programmed control by passing the signal for construction of three transmission cells, in the order of one right pointing ordinary state, then two up pointing special states, with sufficient relative delay. This mechanism avoids acceptance of multiple pulses by auto-initialisation circuits, and is well served by the delay associated with the reading of an external description; this is an example of the value of long delay within a von Neumann cellular automaton.

The portal cells which transfer pulses to auto-initialisation circuits are indicated by a box around the cell in the figure. The single line box indicates the two points of acceptance of the first start pulse. The second start pulse is accepted by the cell with a double line box. The third start pulse is accepted by the cell with the thick line box. This third pulse is then input to the shaded cell. Upon accepting a pulse, each stage puts signal to clocks, and to the adjacent special constructor, ensuring that the next phase will receive the next pulse of auto-initialisation, and that only three such pulses are accepted; the auto-initialised *rtco* as given in the figure is not reconfigurable a multiple number of times. Thus is start signal delivered to the six auto-initialisation circuits. All further input to the organ, after a short delay following auto-initialisation, is crossed.

Three different auto-initialisation circuits are used in the *rtco*, one employed for the outside four clocks, and two for the one internal clock. Auto-initialisation occurs in three phases, each phase started by a separate pulse. Phases one and

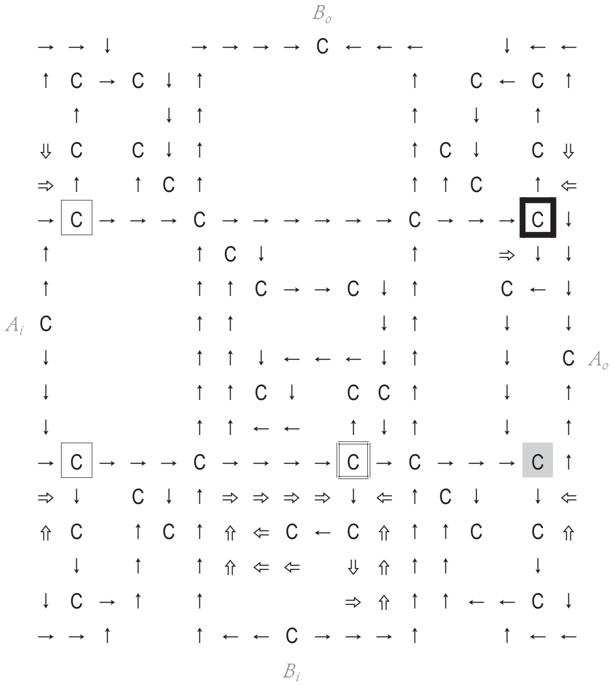


Fig. 17. The *rtco*, configured for auto-initialisation. The minimum start signal for this organ is $\mathtt{10}^{13}\mathtt{10}^{32}\mathtt{1}$, which is applied once to A_i. The organ begins crossing signal 30 clock ticks later. This organ can be started at the time of construction; there is no start signal propagation. All signal crossing paths impose equal delay. The organ has 18×18 dimensions. This configuration can be marginally reduced in area.

three initialise the outside clocks, while phase two initialises the central clock. The auto-initialisation circuit for outside clocks produces two signals, a pulse, and the $\mathtt{1}\overline{\mathtt{01}}$ signal. The pulse is passed to the special transmission path constructor, which annihilates the auxiliary path, closing the signal port, and protecting the auto-initialisation circuit from being started multiple times. The purpose of the constructor in phase two is a bit more complicated. The pattern of alternating bit values in the signal carried by the *rtco* must be observed at all signal crossing elements. Further, the length of the signal paths surrounding the central clock must yield a signal delay of even time. From a design point of view, it is better to have shorter signal propagation delay (and correspondingly fewer parts), and therefore it is better to remove one confluent cell than to add seven others; the confluent cell must change to the right ordinary transmission state. Hence, there is no need to delete the auxiliary path used to initialise the central clock. The post-auto-initialised *rtco* is shown in Fig. 18, with shading to highlight

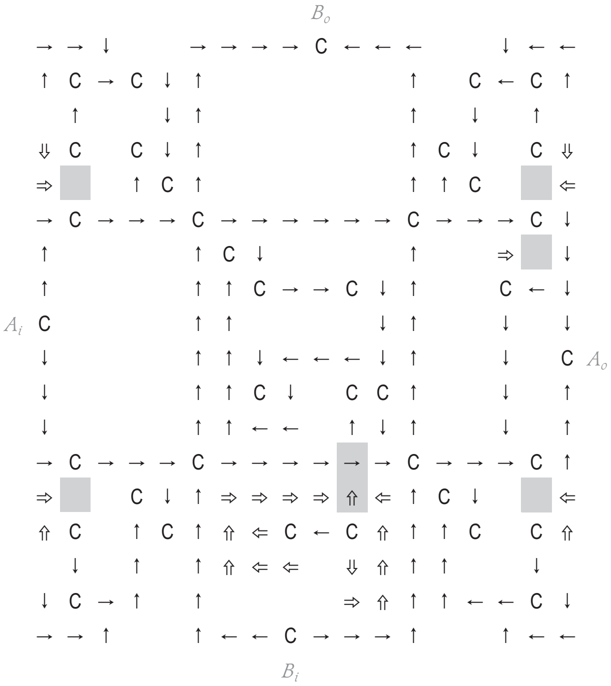


Fig. 18. The *rtco*, after reconfiguration by auto-initialisation. Cells with altered states are shaded. Note that all auto-initialisation circuits are isolated from organ signal paths.

those cells having altered state. Isolation of signal path from auto-initialisation circuits is clear.

An important part of auto-initialisation is that start signal not be propagated beyond the auto-initialised organ. For some organs, it may be necessary to construct special configuration to prevent premature signal propagation. The design of the output ports of the *rtco* is sufficient to prevent start signal propagation. Hence, the auto-initialised *rtco* can be employed fully started, even as the configuration of which it is a part remains under construction. As we see, construction of the auto-initialised *rtco* is substantially easier than is any means of constructing the non-auto-initialised *rtco*, as given by Gorman.

Clearly, any constructor capable of constructing the *rtco* by means other than auto-initialisation carries within an albatross.

5 The Mukhopadhyay crossing organ

The *mco* is a perfect signal crosser composed of three identical *exclusive or* organs, each being larger and more complex than is even the auto-initialised *rtco*. The *mco* is well described in Buckley and Mukherjee [1], where the reader should turn for details. Yet, the configuration is not therein presented in its entirety. We see the complete non-auto-initialised *mco* in Fig. 19, and observe an increase in the size and number of signal crossing elements, a corresponding increase in the number of signal paths, the more complex bit processing mechanism, and the nine-fold increase in clock count, as compared to the *rtco*. The auto-initialised *mco* is easily obtained by substituting the three component *exclusive or* organs of Fig. 19 with the corresponding *exclusive or* organ described in Fig. 5 of Buckley and Mukherjee [1, pp. 401], and adjusting signal path delay between these organs, as necessary. The start signal is akin to that used for the *rtco*, though not a pulse. Rather, as the *mco* is started in a staged fashion, the signal comes in three packets, with suitable relative delay between the packets; each *exclusive or* organ is started by separate signal. Packets are five bits in duration; `11111`. Like the auto-initialised *rtco*, auto-initialised *mco* design prevents the premature propagation of signal.

Clearly, the *mco* is no solution to *rtco* construction difficulties. Neither is it space efficient. The non-auto-initialised *mco* is much larger than is the *rtco*, auto-initialised or not, being about one third the size of a known vNCA self-replicator. The *mco* is also less elegant than the *rtco*, and expresses a bit of Rube Goldberg in its design. It is a mechanical solution to signal crossing, which incorporates construction as part of its operation, being therefore a self-modifying device, much befitting of von Neumann. The operation of the *mco* is altogether an order of magnitude more complicated than is the *rtco*. Consequently, the *mco* is not a practical signal crosser, even if more optimally designed. More efficient solutions to signal crossing exist, including the *rtco*. The unsuitability of the *mco* to practical application does not extend to the constituent *mi* signal inverter.

6 Ideal signal crossing

We close our review of signal crossing with consideration of ideal mechanisms. In as much as a system of cellular automata is defined, in part, in terms of states and transitions between states, ideal signal crossing is not obtainable within vNCA. Yet, by careful alteration of these two characters, one may obtain an analog which functions largely as vNCA, and provides for ideal signal crossing. Nobili and Pesavento [14] obtained this result, to produce what we name Nobili cellular automata, NCA. Their changes endow the confluent state with ideal signal crossing, at the cost of function as a two input, two output *logical and* operator, and gives confluent states having inputs but no outputs the ability to indefinitely store data. In all other respects, NCA are identical in behavior with vNCA. Our concern here is with the signal crossing function, represented by the two input, two output conformation, and any benefit thereby obtained.

Clearly, the availability of ideal signal crossing eases the burden of a designer, reducing the contingency load for signal crossing to that of positional relation relative to other component organs. Nobili [13] reports this as his purpose, simplifying his analysis of universal construction. Nevertheless, we argue that the chief benefit of ideal signal crossing within NCA is configuration size minimi-

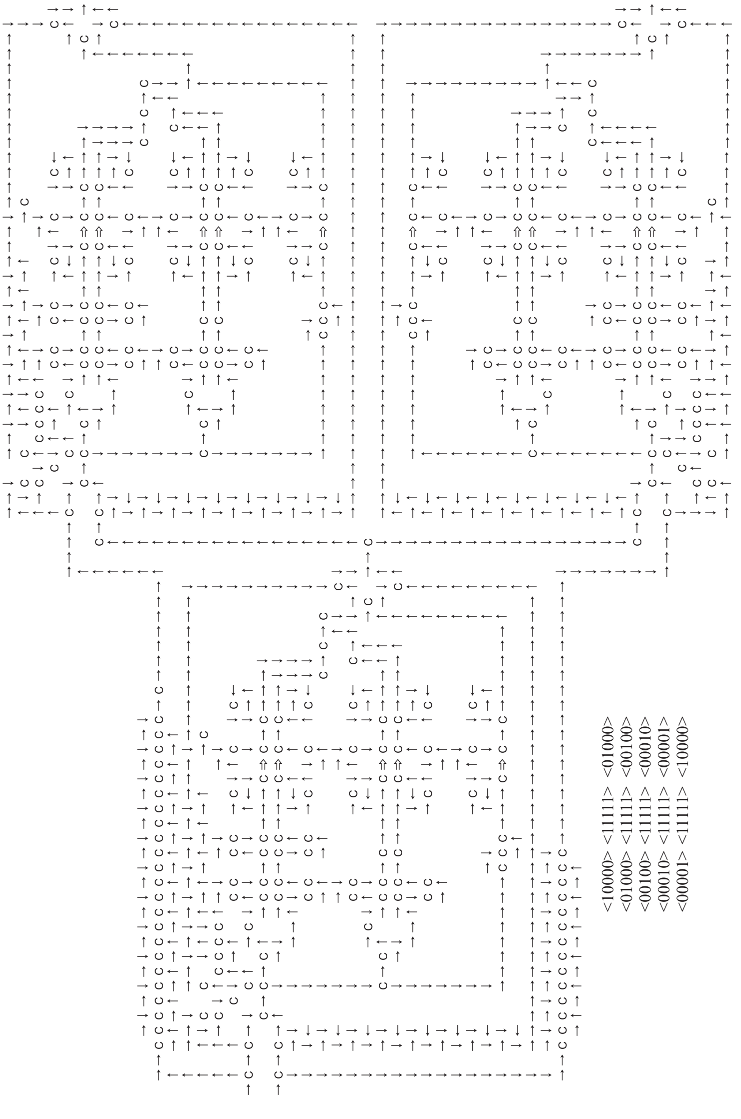


Fig. 19. The non-auto-initialised Mukhopadhyay [12, 1] crossing organ, *mco.*

sation, versus functionally equivalent vNCA configurations. Indeed, in the case of one example, we use Nobili's signal crossing extension to demonstrate a self-replicator for vNCA, via the configuration of its NCA isomorph, replacing ideal signal crossing with the service provided by the 4×5 *cc.*

The parallel relationship of the current paper with the work of Nobili and Pesavento [14, 15], joint and several, gives us pause to consider. Nobili's [13] frank admission of introducing ideal signal crossing as a means to simplify other cel-

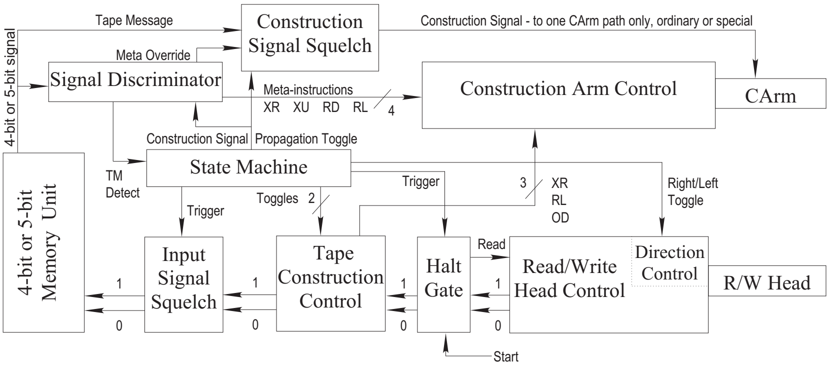


Fig. 20. The standard architecture of a self-replicator for Nobili and von Neumann cellular automata. All internal paths carry pulse signal only. A total of ten signal path intersections are indicated; all pulse signal.

lular automata investigations points up the important pre-requisite of thorough understanding of signal crossing to the construction of a vNCA self-replicator. Pesavento [15] provides a design which, while clearly suggestive, proves insufficient as to self-replication within NCA, and by extension vNCA, even as it is a capable NCA constructor; the configuration is unable to construct one cell of its corpus. The relevance of this point concerns the development of a standard architecture for self-replicators within NCA and vNCA, which originated with our analysis of the Pesavento [15] design. The standard architecture is shown in Fig. 20.

Most of the example self-replicators presented herein conform to the standard architecture, if with occasional inexactitude. Yet, the implied sequence of events and the flow of information suggested in the standard architecture apply even where the architecture of a specific self-replicator differs; the last self-replicator discussed operates similarly, though it has a completely alternate architecture. Clearly, derivation-from implies correspondence with the Pesavento [15] design, and this is true also of the design presented jointly by Nobili and Pesavento [14], a general constructor over the first quadrant, implemented for vNCA. Like the Pesavento [15] design, this jointly developed configuration is insufficient as to self-replication.

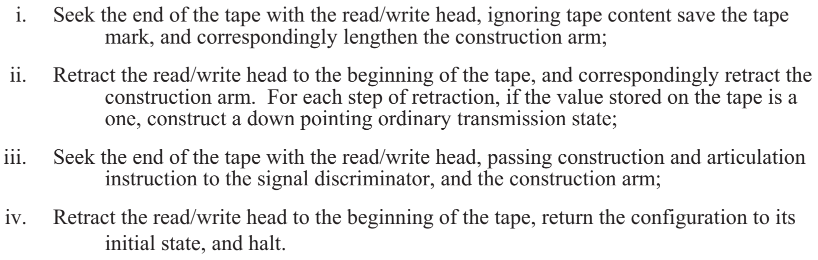
7 Signal crossing in self-replicators

As we have said, it is generally understood that signal crossing within vNCA is an important problem in the design of complex automata like self-replicators. We see in the standard architecture a few examples of signal crossing. Many others are hidden within the organs; the controls for the construction arm and read/write head are particularly rich in crossed signal. Having examined the construction and behavior of the two available classes of signal crossing organs, the discussion now properly turns to application of signal crossing within the self-replicators of NCA, and their vNCA cousins. We begin with analysis of the systematic behavior of the standard architecture, especially communications between the organs, and develop an understanding of how those communications coordinate self-replication. We then present four self-replicating configurations, giving a detailed presentation of three of these, noting special characteristics and showing their correspondence with the standard architecture. Discussion then turns to a novel mechanism of self-replication, termed partial construction, which is compared with other published cellular automata self-replicators. Owing to their size, the images of these four configurations are shown within an online manuscript, located on the world wide web at the URL listed at the end of the Comments section. Each figure is referenced in the foregoing text by mention of the catalog, and the page number upon which the corresponding configuration is shown.

The primary task of a self-replicating automaton is the conveyance of construction and articulation instruction from information source to information consumer; the tape and the constructor. Self-replicator design is then the task of selecting the mechanism of information conveyance, which mechanism consequently determines signal crossing requirements; the environment imposes the requirements for construction. The process proceeds through physical implementation of a mapping, each instruction producing a single pulse routed to a particular pulser, a translation from instruction code to construction and articulation signal. Instruction code translation is minimised by use of construction signal as instruction code, partitioning these from meta-instructions, like those directing construction arm articulation and the *tape mark*, this having minimal use demarcating tape end-points. Construction signal is suitable as instruction code, being short in length. Quite the opposite applies to articulation signal. Upon this process of instruction conveyance and translation are built those acts that define the steps of automaton self-replication, tape replication and configuration construction, acts reflected in the algorithm of the standard architecture.

We see in the standard architecture that construction and articulation signal is delivered to the construction arm and its control from three sources, each source governed by one or more pulses. These pulses originate within the state machine, which services a closed cycle of four states, corresponding to the four behaviors we argue necessary to self-replication; tape replication and configuration construction including replicant triggering at **Start**, both mentioned above, and two rewinds of the tape, to include returning the configuration to its initial condition, or at least evolved to a configuration from which it can produce further replicants, being restarted for each at **Start**; algorithms are expected to halt. These behaviors are listed in Fig. 21, in typical order. Tape replication corresponds with state (ii), and configuration construction with state (iii); alternative orderings are equally effective for self-replication. This sequence of behaviors is the output of each example self-replicator; they are restartable.

As to necessity, others may disagree with us regarding a return to initial condition; see for instance the configurations of Morita and Imai [11]. One may cor-



i. Seek the end of the tape with the read/write head, ignoring tape content save the tape mark, and correspondingly lengthen the construction arm;

ii. Retract the read/write head to the beginning of the tape, and correspondingly retract the construction arm. For each step of retraction, if the value stored on the tape is a one, construct a down pointing ordinary transmission state;

iii. Seek the end of the tape with the read/write head, passing construction and articulation instruction to the signal discriminator, and the construction arm;

iv. Retract the read/write head to the beginning of the tape, return the configuration to its initial state, and halt.

Fig. 21. The behaviors of a von Neumann self-replicating cellular automaton. The ordering shown corresponds to the sequence observed in the example self-replicators. Other orderings are effective.

respondingly neglect state (iv), to taste, eliminating the restart ability property from resulting configurations, which may in consequence be marginally smaller. We leave for another time any ensuing debate.

It is important that we explain to some detail the operation of some standard architecture organs but, not to the finest detail. Nor need we describe all organs. In particular, the construction arm and read/write head are well discussed elsewhere in the references. We devote little space to these organs, as their overall operation does not change between examples, even if configuration specifics do. For instance, in most published examples, the construction arm is organised with special transmission states positioned above ordinary transmission states; our examples exhibit this and the reverse ordering; control signal differs between the orderings. Yet, both example organs serve the same function and command set, those listed in Fig. 2. Our discussion need not include details of signal which is specific to particular variations of these organs.

Still, the interaction of these two organs with other automaton configuration is important to the discussion. For instance, the read/write head operates independently of all other organs, and drives all other automaton behavior; this is also true for the Nobili and Pesavento [14, 15] designs. Each pulse delivered to **Read** causes the reading of a single bit from the tape, any repair owing to destructive tape reads, and positional adjustment of the read/write head and signal return paths, these bounding the top and bottom edges of some portion of tape. Self-replicator behavior terminates upon interruption of this cycle, which is the purpose of the *halt gate*, at the direction of the state machine.

The reading of successive bits from tape must be delayed by time sufficient to allow completion of the internal operations suggested above, and is signaled by sampling the **0** and **1** input signal lines via *logical or*, the resulting pulse being then passed to **Read**, and other places internal to the read/write head control. Several hundred clock ticks between the reading of two adjacent bits of the tape are not unreasonable. Indeed, it is often necessary substantially to increase this delay, facilitating signal synchronisation; it is expected that instructions be fully processed before the first bit of the next instruction is read from tape.

Finally, tape bit location adds significantly to this minimum; the reading of data from tape is overwhelmingly the largest consumer of time for von Neumann self-replicators. On average, a tape read signal travels the length of the tape between the start of reading a bit (the sending of signal) and its completion (the receipt of signal), this measure being within the error bars expected of a physicist.

As a practical matter, the expectation for full processing of one codon before starting to read the next from tape is relaxed, providing opportunities for optimisation over time, however marginal. As with most optimisations, this comes at cost; at the moment when *tape mark* input is fully processed by the state machine, the read/write head has already begun reading one cell past the end of the tape. In addition to need for a change in the direction of tape traversal and a repositioning of the read/write head along the tape, the resulting input signal, the read of a one or a zero, needs to be culled from the **0** and **1** input signal paths. The input signal squelch, *iss*, culls this unwanted signal. The overshoot behavior is observed for all traversals of the tape by the read/write head, in all examples, and in the Pesavento [15] design; the 29 state design of Nobili and Pesavento [14] cannot rewind its tape, and only constructs.

Service of those bits read from tape is obtained by two sub-configurations. One consists of the memory unit, signal discriminator, construction signal squelch, *css*, and state machine organs, which together service codons directing construction and articulation, and machine state; this is the general constructor. The other is the tape construction control, *tcc*, which directs tape replication by individually serving each bit as it is read from tape. The memory unit groups tape data into codons which the signal discriminator divides into three categories, governing subsequent distribution of those codons. The *css* is triggered by detection of meta-instruction, preventing delivery of those codons to the construction path of the construction arm, and by the state machine. *Tape mark* detection yields pulse signal to the state machine; two detections yield a transition of state. All other meta-instructions govern construction arm articulation, and are delivered to the construction arm control.

Tape replication is the simpler function. First is the determination of tape length and the corresponding extension of the construction arm. Next follows a rewind of the tape and corresponding retraction of the construction arm as it constructs the tape replicant. Construction signal is serviced before articulation signal. This process is driven by the read/write head. In state (i), the *logical or* of signals **0** and **1** are delivered to construction arm input **XR**. In state (ii), the *logical or* of these same signals is delivered to construction arm input **RL**, as **1** is delivered to input **OD**.

During the transition to state (ii), pulses are sent from the state machine to the *iss*, culling the unwanted input bit; to the *tcc*, squelching signal **XR** and permitting propagation of signals **RL** and **OD**; and to the read/write head, directing a reversal of direction for tape traversal. Distribution of these signals entails some signal crossing. Being thorough, we mention the existence of some subtle timing relations between the acts associated with read/write head behavior, which coordinate to ensure that reading of the tape during state (ii) begins with the last (right-most) bit. Though reader understanding of these details is not relevant to the purposes of our discourse, they will prove important to any reader who undertakes a painstaking review of the internal operations of examples.

State (ii) is the reverse of state (i), save for the delivery of signal **OD** to the construction arm, yielding replication of the tape. Upon detection of the second *tape mark*, the previously described conditions for reversal of tape traversal direction exist, like read/write head overshoot, and are correspondingly serviced (we do not mention these conditions in the discussion of subsequent state machine transitions, even though they may apply). In addition, pulses open the *css*, permitting the flow of construction signal to the construction arm, and de-asserting the signal discriminator squelch, permitting the flow of articulation signal to the construction arm control. The configuration transitions to state (iii).

R/W Head	for each bit of tape input, extend the read/write head; read/write head is positioned over the first, left-most bit of the tape; reading occurs, left to right
Memory Unit	awaiting receipt of first codon bit
State Machine	in state one, awaiting detection of the first tape mark
Signal Discriminator	set to suppress propagation of signal to the construction arm
Input Signal Squelch	set to open; signal is allowed to propagate
Construction Signal Squelch	set to closed; signal is not allowed to propagate
Halt Gate	set to open; signal is allowed to propagate
Tape Replication Control	for each bit of tape input, transmit XR signal to the construction arm

Fig. 22. The initial state of all example self-replicators. The construction arm is not configurable.

Standard architecture construction is best understood in terms of the inter-operation of the organs. The initial state of standard architecture self-replicators is shown in Fig. 22. The sequence of events for self-replication begins with injection of a pulse at **Start**. This starts a continuous reading of the tape; reading does not stop until action is taken by the state machine. For states (i), (ii), and (iv), the *css* is asserted by the state machine, preventing the propagation of codons to the construction arm. Though the signal discriminator is always operational, an internal squelch is asserted on output to the construction arm control; discriminator signal is propagated to the construction arm during state (iii) only. Output from signal discriminator to state machine is propagated during all states; there is no squelch on signal **TM Detect**. During state (i), the *tcc* emits only signal **XR**, and the *iss* is de-asserted, permitting propagation of all incident signal. The *halt gate* is also de-asserted, permitting continuous automaton behavior.

The memory unit services codons during all states. For all states, codons move from memory unit to signal discriminator, and with short delay to the *css*. If the codon represents a meta-instruction, the *css* is signaled to prevent codon propagation to the construction arm. Signal representing identified meta-instructions are delivered to corresponding inputs of the construction arm control, save that for the *tape mark*, which representing signal enters the state machine. For any codon, only one signal is put to the construction arm. Again, override signal from the state machine further limits the propagation of the foregoing signal to state (iii).

Construction ends with a transition to state (iv). The final state returns the configuration to its initial condition, ready to receive another pulse at **Start**, and capable of repeating the process of self-replication. The final act is repositioning of the read/write head concomitant with detection of the second *tape mark*. In this one case, the state machine issues signal to the *halt gate* during transition to state (i), squelching the **0** and **1** signal paths, halting the read/write head, and consequently the configuration, completing self-replication.

The configurations of example self-replicators generally correspond to that given in the standard architecture, to include relative placement of organs, and follow directly.

Nobili pulse corpus With the overview of standard architecture behavior complete, we now proceed to describe our three example self-replicating configurations. For each example, we present such detail as is necessary to understand operation of each organ, with special emphasis placed upon signal crossing requirements. We will also take note of variations to organ design, such as are necessitated by differences between NCA and vNCA. Our four example self-replicators demonstrate a progression; the pattern of states for the initial condition is given for each in a corresponding figure.

The first example configuration demonstrates that self-replication within NCA is not dependent upon the service of packet signal; that ideal and indeed even perfect signal crossing is overkill. The second configuration demonstrates the expected size reduction obtained by incorporating the service of packet signal, while at the same time demonstrating that service of packet signal implies no crossing requirement. We name these two NCA examples according to the signal they service; the pulse corpus and packet corpus configurations. The conversion of these two self-replicators from NCA to vNCA is then demonstrated. For the first resulting configuration, we present just the dimensions; generation of the configuration is left as a reader exercise. For the second conversion, the configuration is given, and we argue it to be architecturally the smallest possible vNCA self-replicator.

Some of the standard architecture organs are common to all the examples, and these often without substantial change. Of note is the state machine organ, which was derived from the work of Pesavento [15]. The state machine and *iss* have few differences between examples. The *halt gate*, found in all examples without alteration, is shown in Fig. 23. Other organs exhibit substantial variation between the examples; in the first example, the memory and signal discriminators are replaced. Variations between the *iss* for NCA and that for vNCA relate solely to signal crossing requirements; instead of ideal signal crossing, the 4×5 *cc* is employed. Like the *iss*, differences between NCA and vNCA expression of the *tcc* relate solely to mechanisms of signal crossing. This is not so for other organs; the

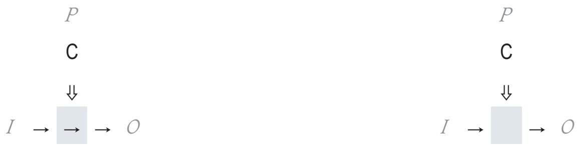


Fig. 23. The *halt gate*, in the closed and open conformations, from left. This normally closed switch is opened by a pulse at P, and reset by a pulse at I. When closed, all signal is passed to O.

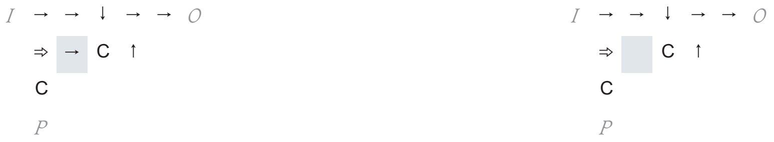


Fig. 24. Construction signal squelch, in the closed and open conformations, from left. When the switch is open, signal passes from I to O. Multiple inputs to P are combined by a *logical or* operator.

differences between the example memory units exceed alteration in component signal crossing mechanisms. Some configurations employ multiple *css* organs, shown in Fig. 24, linked together by a single control signal. Like the *halt gate*, the *css* appears in all examples without change, whether singularly or in multiple.

The *halt gate* operates by annihilating one right pointing ordinary transmission cell of the signal path that post-*logical or* leads from the **0** and **1** input signal lines to the **Read** input of the read/write head control. This ordinary transmission cell is then reconstructed by passage of a final signal from the combined input signal lines; a pulse is identical with **OR**, and an ordinary transmission state pointing to a ground state is a constructor. The side effect of reconstruction is that the signal is not passed further, and reading of the tape stops.

The *css* operates by construction and annihilation, to control a valve on the signal flowing through, and not to alter, a signal path. Again, this control is effected by agency of a side effect. An ordinary transmission state serves as the control on a confluent state, through which signal passes; when the ordinary transmission state is present, the confluent state carries no signal. The confluent state performs a *logical and* upon its inputs. The logical operator blocks signal, as one input has a constant value of zero. Upon annihilation of the ordinary transmission state, no longer is a *logical and* performed upon inputs, and signal is not blocked. The side effect is the engagement of the logical operator.

The *tcc*, shown in Fig. 25, is a passive configuration employing two signal crossing organs. The three generated signals are connected to the construction arm control, with an intermediate signal squelch on each, subject to control of the state machine; the signals are produced for each read of the tape. It is the careful selection of moment to permit propagation which makes sense of these signals; they would produce chaotic results when applied all together during any

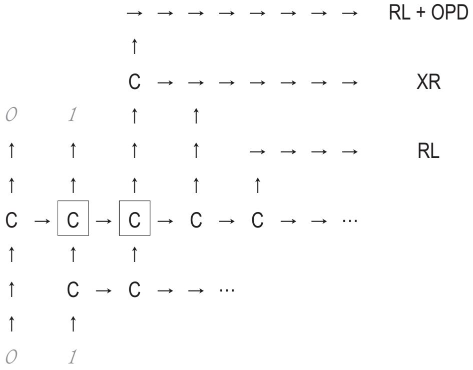


Fig. 25. The NCA tape construction control, *tcc*, is a grouping of three signals, these being derived from combination of the **0** and **1** signals emitted by the read/write head. The state machine mechanisms governing propagation of these signals is not shown. Component signal crossing organs are shown boxed.

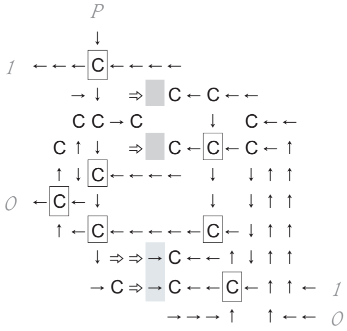


Fig. 26. The two channel input signal squelch for NCA, shown in the open conformation. Pulse signal passes from input to output when this organ is in the open conformation. A pulse at *P* toggles the open/close state of the organ, moving the pair of right pointing ordinary transmission states from the position of the lightly shaded cells, to the position of the darkly shaded cells. The first input resets this state, returning the right pointing ordinary transmission states to the position shown. The boxed confluent states are ideal signal crossing organs.

automaton state, and corrupt the process of general construction during state (iii), among other behaviors.

The *iss*, shown in Fig. 26 for NCA, is a pairing of dual-path valves, the operation of each path-pair being exclusive to the other. Further, one permits signal passage while the other does not. The selection of which path-pair does, and which path-pair does not, pass signal is determined by signal accepted from the state machine. The output of one path-pair, combined by *logical or*, is input

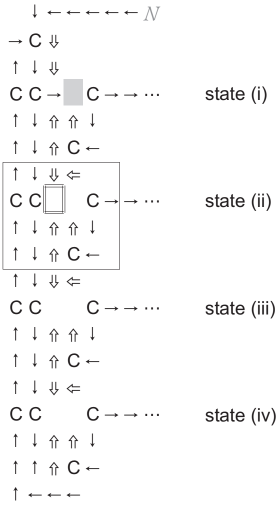


Fig. 27. The state machine, configured for Nobili cellular automata, includes one signal crossing organ. The state machine is initially set into the first state, and supports a total of four states. Each state corresponds to one complete traversal of the tape. Upon four traversals of the tape, two from beginning to end, and two from end to beginning, the state machine is returned to the configuration shown here. The boxed region with a single line identifies the cells which define the second state. The double-boxed cell is described in the text. This organ is derived from the state machine given by Pesavento [15].

at P, providing a one-shot bit bucket; subsequent signal propagates through the other path-pair. Normally, signal flows from the *iss* to the memory unit; the normal signal paths. Upon direction of the state machine, *iss* configuration is altered, redirecting flow to the alternate pair of signal paths, thus facilitating *iss* reset by the next P incident signal originating in the read/write head.

The state machine for the pulse corpus includes just one signal crossing organ, as shown in Fig. 27. The larger single-line box in the figure shows the sub-configuration responsible for all features of machine state (ii); its entry, maintenance, and exit. The machine state indicator is implemented as a sequence of five cells. The rightmost and left two cells are confluent states. The other two cells, sandwiched between confluent states, alternate between ground and right-pointing ordinary transmission states, indicating the current state of the machine. A machine state is entered when a right-pointing ordinary transmission state is constructed in the left of these two sandwiched cells. A right-pointing ordinary transmission state is constructed in the right cell upon **TM Detect** being passed to input N. A second such signal triggers transition to the next machine state.

Machine state transition brings annihilation of both right-pointing ordinary transmission states of the current machine state indicator, and the propagation

		Low Order Bits			
		00	01	10	11
	00	OR	SR	GG/RL	CC
High	01	OL	SL	XR	
Order	10	OU	SU	RD	
Bits	11	OD	SD	XU	TM

Fig. 28. The tape instruction code for the pulse corpus NCA self-replicator. Retraction of the construction arm to the left yields default construction of the ground state. Use of unassigned codes yields no construction arm output.

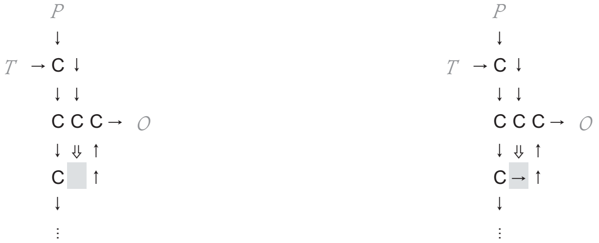


Fig. 29. The crossbar connection unit, *ccu*, unset and set, from left. A pulse at T toggles the construction and annihilation of the indicated ordinary transmission cell, thereby closing and opening the switch. A pulse at P is propagated to output O when the switch is closed.

of pulse signal to the next state, which constructs a right-pointing ordinary transmission state in the left cell of the next machine state indicator, shown with a double-line box in Fig. 27. The machine thus moves into state (ii). Pulse output of the state machine is directed to various signal lines, providing control to the other organs of the configuration. We consider these signal lines as separate from the state machine. One may expect that the network of signal lines from the state machine intersect many signal crossing organs.

The pulse corpus differs most from the standard architecture in that the signal discriminator and memory unit are combined into one sub-configuration; the instruction decoder. The instruction decoder combined with state machine constitutes the supervisor of the self-replicator, and is built largely of signal crossing organs; a total of 258 occur within the instruction decoder, and 436 overall. Further, the size of the instruction decoder is approximately 60% the area of the self-replicator. The instruction code for the pulse corpus is shown in Fig. 28. The reader should note that retraction of the construction arm implies the construction of two ground states. Thus, these operations are combined into a single instruction of the code.

The instruction decoder is organised into three sections, which in total act as a 4-to-16 demultiplexer. This demultiplexer is the core functionality which facilitates the translation of instruction code into construction and articulation

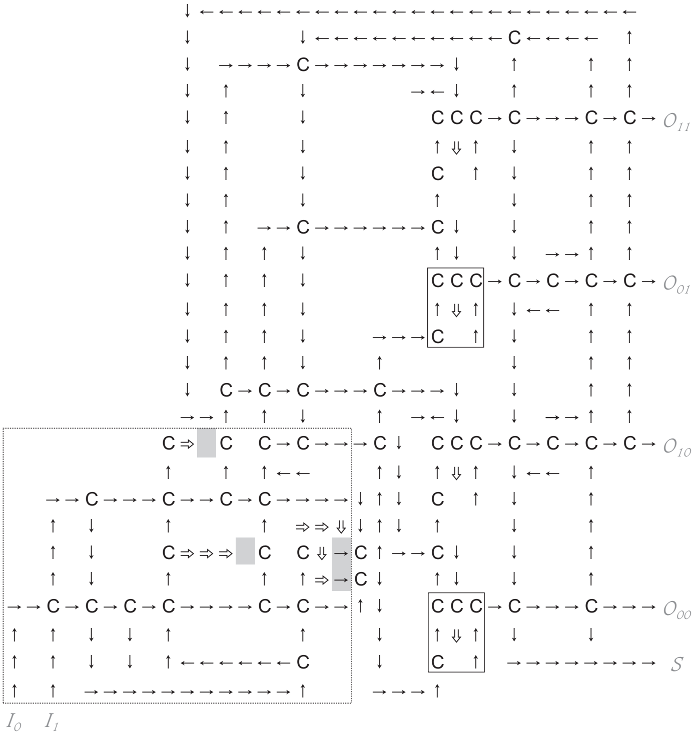


Fig. 30. A 2-to-4 demultiplexer, built of four crossbar connection units. This organ converts two input pulses into a single one-of-four output pulse. The region bordered with a dashed line distinguishes between the first and second input pulse. The regions with a solid line boarder are those crossbar connection units that will be set if the first input pulse represents a zero bit on the tape. The other two crossbar connection units will be set otherwise. The inputs are labeled I and the outputs O, each with subscripts to indicate related logical value; for outputs, subscripts correspond with the input pulse pattern that yields output at the port, the first bit being leftmost in the subscript. S serves to distinguish input pulse one from input pulse two within the 4-to-16 demultiplexer, which follows downstream of the 2-to-4 demultiplexer.

signal. The first section gates alternating input signal between two input paths. These two paths then serve as the inputs of a 2-to-4 demultiplexer, which outputs are then input to the 4-to-16 demultiplexer. Like the memory unit, this organ accepts pulse incrementally, and it is only upon input of the last pulse that output is produced. Each pulse of input brings some kind of reconstruction. The complete operation cycle of every sub-component of the instruction decoder serves as its own reset.

The two demultiplexer sections share a common design, based upon the crossbar connection unit, *ccu*, shown in Fig. 29. The 2-to-4 demultiplexer employs a set of four *ccu* organs, while the 4-to-16 demultiplexer employs a set of sixteen *ccu* organs. These sets are logically organised into squares, 2×2 and 4×4. Operation of the *ccu* is in response to three ordered pulses, the last being feedback on organ output. The first such pulse, input at T, yields construction of a right-pointing ordinary transmission state in the shaded cell of the figure. The second pulse is accepted at P, and propagates to the confluent state at the left of the shaded cell. All *ccu* organs in a row receive the same inputs at T, and the *ccu* organs of columns receive the same inputs at P. The 2-to-4 demultiplexer is shown in Fig. 30.

In addition to connection with an input of the construction arm control, each of the output signals from a row of *ccu* organs is combined by *logical or*, giving feedback to the T input of those same *ccu* organs. This signal resets the states of the *ccu* organs, by destructing the component right-pointing ordinary transmission state. A similar mechanism applies to the 2-to-4 demultiplexer. The 4-to-16 demultiplexer is identified by the 4×4 *ccu* array at the center to top of catalog page 4.

In our discussion of *rtco* construction, we mentioned that the simplest programmed control is the one which makes no decisions on its own; the prime example being compression of the tape. This first example self-replicator is not so simple, for **RL** signal to the construction arm control is concomitant with construction signal to the construction arm. That is, this configuration assumes construction arm retraction for each act of construction. This automaton behavior provides a means to optimise the configuration description versus tape length, and so reduce the time to self-replication. Yet, this also means that the configuration is larger than it need be, for functionality comes upon the heals of configuration. We expect that a description of this configuration viable for self-replication will be no greater than nine times configuration size. The area of this first example, a Nobili self-replicating cellular automaton, is 220 by 109, or just under 24K cells. The number of cells within the perimeter of the configuration is roughly 18K. Further, many of these endo-perimetric cells are in the ground state; it is likely the footprint can be significantly reduced. We estimate that the minimal size is about 12K cells. Size minimisation of this configuration is an effort left to readers.

We have thus shown that NCA self-replicators exist for which the ability to cross packet signal is unnecessary. Our attention now turns to demonstration of the size reduction benefit expected from the addition of service for packet signal within NCA self-replicators.

Nobili packet corpus The packet corpus differs from the pulse corpus chiefly upon the variation of instruction decoder versus memory unit and signal discriminator, an increase of codon size to five bits, and the service of packet signal, to include its crossing. The addition of packet signal and its crossing serves two purposes. We wish to demonstrate the benefit of packet signal crossing to the minimisation of configuration size. We wish also to demonstrate the maximal benefit obtained from ideal signal crossing. As this second example behaves

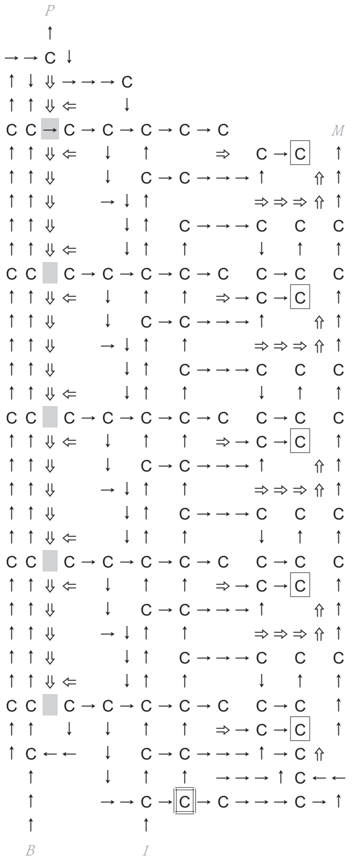


Fig. 31. The NCA packet corpus memory unit, a first-in, first-out pulse to packet register. Inputs are the signal **1** and the *logical or* of signal **0** and **1**, labeled B. The shaded cells indicate the mechanism of counting bits of input, and triggering their output, which is provided in two signals, a pulse P (presence) and the packet signal M (message). Boxed cells are memory states; note the lack of output states in each neighborhood. This organ derives from the memory unit given by Pesavento [15].

otherwise much as the first, we shall limit figures to those which demonstrate unfamiliar organs. We should mention, by way of full disclosure, that development of the packet corpus was in direct response to limitations of the design given by Pesavento [15]; that the corpus is not of a self-replicator. Each configuration accepts descriptions represented in a common tape code, and tapes contain three *tape marks*, not two. Also, the state machines represent more than four states; six for the Pesavento [15] corpus, and seven for the packet corpus. Consequently, neither configuration is minimally sized; smaller NCA self-replicators exist. Yet, for the packet corpus, these failures at size optimality are not of great extent; the Pesavento [15] design compares less well. Hence, we may by this second example obtain a good first approximation of the size reduction benefit of packet signal service within NCA.

The design of our memory unit, shown in Fig. 31, being approximately 45% smaller, is derived from the memory unit of Pesavento [15]. Both units accept five bits per codon, and automatically emit the codon upon its complete acceptance, readying for input of the next codon. Though unused, the Pesavento [15] design includes configuration for the latching of signal, allowing held signal to be output in multiple; our design eliminates this feature. The position of the right-pointing ordinary transmission state in the shaded cell indicates the position of storage for the next bit of input, located at the end of the adjacent right pointing signal path; the organ is configured to accept the first bit of input. This indicator is paired with another, represented by the lack of right-pointing ordinary transmission state. This lacking state permits signal delivery to the memory states, which are shown as boxed cells in the figure.

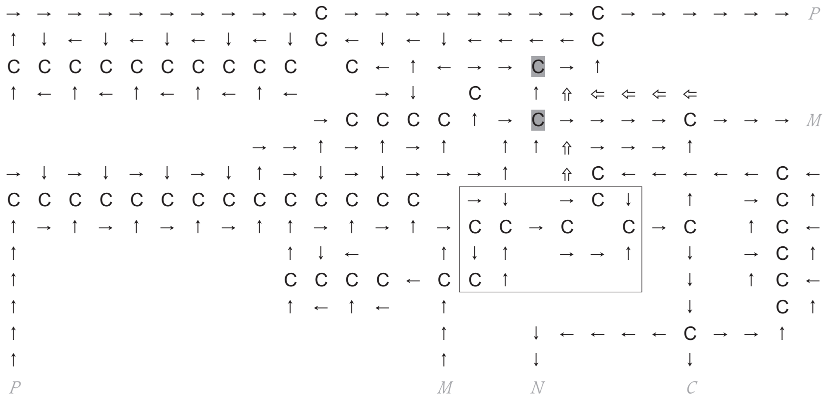


Fig. 32. The tape-mark service stage of the NCA multi-bit corpus signal discriminator. Integral constructors are built of special transmission paths. The shaded cells are gates, which alter state between a crossing organ and a two input, one output *logical and* operator. The majority of non-ground states of this organ synchronise signals P and M at output.

Input to our memory unit comes in two signals; a pulse B indicating receipt of signal, and the **1** signal. If signal is received but it is not **1**, then a simple increment of pointer to storage cell suffices; the pulse increments the storage cell pointer, along with the paired permission indicator. If **1** is received, then it is placed in the storage cell, and the pointer increments. Output from the memory unit is triggered by the wrap-around of the storage cell pointer; B is propagated to the output control sub-configuration. The stored data is released last memory cell first; delay in the output signal path versus delay in the output control sub-configuration ensures that the bits are properly ordered; most-significant bit first. A right-pointing ordinary transmission cell is constructed just to the right of the storage cell, starting output. As soon as the bit stored in the memory state is emitted, the newly-constructed right-pointing ordinary transmission cell is annihilated. In the two clock ticks between construction and annihilation, the transmission cell conveys the stored bit from memory state to signal output path.

The general signal discriminator is composed of a series of sub-configurations, each consisting of a decoder and a signal squelch. These sub-configurations each prohibit propagation of a specified signal upon its recognition. The signal **TM Detect** is detected first, followed in order by the signals for the **XR**, **XU** and **RD** instruction; **SI** is not implemented in the packet corpus. **RL** is indirectly implemented and so needs no meta-level service; the construction signal 00000 yields no constructed state but, does yield automatic retraction of the construction arm. If none of these four signals is detected, output from the memory unit propagates to the construction arm. The signal squelch applies to both memory output signals, P and M. Figure 32 shows the sub-configuration for detection of the *tape mark*; it is by far the largest signal discriminator sub-configuration.

Two points of interest in this sub-configuration are indicated, by shaded cells and a 4×6 boxed region. The shaded cells operate by side effect, this being the conversion of state, from *logical and* to crossing organ, a behavior described earlier in this text. The boxed region is the **TM** decoder. The decoder emits a pulse upon input of **TM**, which serves to delete the right-pointing ordinary transmission states that are located to the right of the shaded cells. The right-pointing ordinary transmission states are reconstructed after a few dozen clock ticks, restoring the signal crossing capability of the shaded cells. Hence, when **TM** is detected, P and M are not propagated.

The instruction codes of the packet corpus and the Pesavento [15] corpus are given in Fig. 33. Those codes representing the meta-instructions **TM**, **XR**, **XU**, **RL**, **RD**, and **SI** are selected to allow unambiguous recognition with decoders, and organ that accepts signal solely upon the *logical and* of selected bits. This determines the order of service for the meta-instructions. The downstream decoders are served by a common squelch; only **TM** has a dedicated squelch. The common squelch is a bit more complex, and is influenced as well by signal from the state machine; it implements a complex interaction of signals with delay. As downstream decoder operation is similar to that of the *tape mark*, they are not examined in figures. The signal P is used during construction to trigger automatic retraction of the construction arm. Hence, tape code is compressed. This is true of the Pesavento [15] corpus, also.

The NCA packet corpus is shown on catalog page 5. The signal crossing organs are shaded; dark grey identifies 121 crossers that carry pulse signal, and light grey identifies nine crossers that carry packet signal. In the pulse corpus, each instruction is delivered to the construction arm by a dedicated path. With the packet corpus, construction signal is served by a common path. As packet service in NCA is free, there was no special effort to route M around other con-

00000	RL	01000	opr	10000	OPR	11000	SPU
00001	opr	01001	opl	10001	OPU	11001	XR
00010	opr	01010	opd	10010	OPL	11010	SPL
00011	spu	01011	spr	10011	RD (XU)	11011	TM
00100	opr	01100	spu	10100	OPD	11100	SPD
00101	opd	01101	spl	10101	XU (RD)	11101	SI
00110	spu	01110	spd	10110	SPR	11110	CON
00111	spd	01111	con	10111	Not Used	11111	Not Used

Fig. 33. The tape codes of the Pesavento [15] constructor, and the NCA packet corpus self-replicator. These two codes are identical, save the swapping of codes for the **XU** and **RD** instructions; packet corpus assignments are shown in parentheses. Neither packet corpus supports the **SI** instruction. Codes with capitalised names are the expected codes; lower case names are unexpected but supported.

figuration. Rather, the common path is routed as any other path in the configuration. We may therefore infer that the represented reduction in configuration consequential to packet service is near maximal, if not optimal.

The configuration is contained within a 55×118 region; just under 6500 cells. As mentioned earlier, one finds opportunity for a small reduction in the endoperimetric cell count. The upper left region is an obvious example. There may be available means to modestly reduce the size of other organs, too. As demonstrated in the pulse corpus, general construction in the first quadrant is easily supported with a tape code of four bit instructions, and the minimal number of machine states is four. Yet, we do not expect any such reductions in configuration size to be significant. By rough measure we therefore see that packet service reduces configuration size by a factor of two.

The tape description of this self-replicator is about nine times the configuration size; the combined configuration and tape is less than 70K cells.

Von Neumann pulse corpus We now turn to the conversion of the pulse corpus NCA self-replicator, an NSRCA, into the equivalent vNCA self-replicator, a vNSRCA. Conversion is sufficiently simple that details are not given in figures; the conversion is a logically trivial task, even if the mechanics of manual translation are tedious. First, we remind ourselves that input to the instruction decoder is pulse signal, via the two signal lines **0** and **1**. Only one of these signal inputs to the instruction decoder is active at any given time. Thus, for every case of required signal duplication, the subsequent crossing of same can trivially be phased in time such that no simultaneous convergence upon a signal crossing organ occurs.

Similar arguments can be given for all other organs. For instance, the 2-to-4 demultiplexer receives and emits only pulse signal. Operations of the 4-to-16 demultiplexer follows analogously the operation of the 2-to-4 demultiplexer. Further, all packet signal, for operation of the read/write head and for construction, is generated and serviced external to the corpus of this self-replicator. Hence, there is never a necessity that pulse signal simultaneously converge upon a signal crosser within the configuration.

Of course, use of a filter-corrected *cc*, such as that shown in Fig. 12, in place of the ideal NCA signal crosser, yields immediately a vNSRCA. Similar results obtain from auto-initialised *rtco*. In either case, delay would likely be needed to otherwise maintain signal synchronisation but, no such synchronisation is required to obtain the signal crossing service provided by these two organs. The resultant configuration will simply operate at a slower rate as compared to the performance of the NCA version; such a self-replicator is considerably larger than the pulse corpus NSRCA.

Yet, not much more difficult is the replacement of all ideal signal crossers with the 4×5 *cc* presented in Fig. 10. The chief difference is the amount and placement of additional delay. It seems reasonable that bigger organs would entail the greater delay. We may therefore reasonably conclude that the configuration obtained by this replacement, coupled with suitably tight packaging of component organs, yields nearly the smallest possible pulse corpus self-replicator for vNCA. We have made this replacement and compacted the configuration, obtaining a self-replicator that is bounded by a 272×132 rectangle. The perimeter encloses approximately 27K cells, densely packed with non-ground states, giving an upper bound on the size of a vNSRCA.

We have thus shown that vNCA self-replicators exist for which the ability to cross packet signal is unnecessary; that worst case, the crossing of pulse signal is sufficient to vNCA self-replication. The tape description of this self-replicator is about nine times the size of the configuration, to give a combined size of 270K cells.

Von Neumann packet corpus In transforming the packet NCA corpus into a vNCA self-replicator, we elect to demonstrate that packet signal need not be crossed in order that it be served. We also prefer to construct our tapes of four bit codons, and return to the tape code used in our first example. Further, to minimise configuration, we remove the code compression mechanism associated with implied construction arm retraction, and decline to add other configuration. Finally, we change configuration architecture slightly, to include a new operator, *logical negation*; with this, we are able to encode five bit construction signal into a four bit representation upon tape, demonstrating a different form of code compression, and eliminating use of construction signal pulsers with the construction arm control, such as were used in the NCA pulse corpus. While we need not include either of the perfect signal crossers presented herein, we do include *mi*, the Mukhopadhyay [12] signal inverter. The several figures for this transformation show how the replacement of ideal signal crossers with the 4×5 *cc* affect standard architecture organs. The configuration is shown on catalog page 7.

We begin with von Neumann's requirement that a construct be initially passive. Till now, example self-replicator constructs have all strictly observed von

Neumann's requirement for initial configuration passivity, being started only after construction has been completed, and from a cell external to their corpus. Here, we bend this rule, to the extent that auto-initialised organs are started during replicant construction; i.e. the Mukhopadhyay [12] signal inverter. The requirement for passivity until completion of construction may be preserved with but a small change to automaton configuration. Compliance is effected by the inclusion of a signal path from the location of start signal input to the start signal generator of the Mukhopadhyay [12] signal inverter, increasing self-replicator size; this is the configuration that we decline to include. Interestingly enough, restartability will not require that this path include a signal isolation feature; the timing of the start signal relative to all other signals ensures that it will not be propagated past the signal inverter, which itself accepts only one start signal.

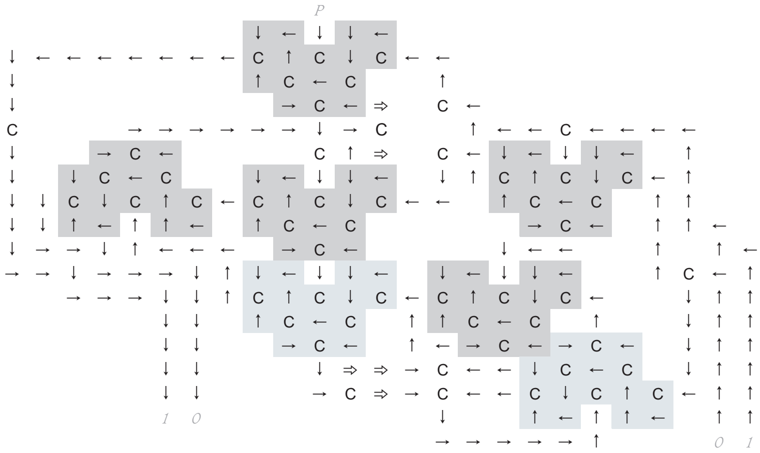


Fig. 34. The *iss* for vNCA, configured with 4×5 *cc* signal crossers; these are shaded to ease identification. Operation of this organ is identical to that of the organ given in Fig. 26.

Figure 34 shows the *iss*, with seven component 4×5 *cc* units; it is otherwise identical in form and function with the *iss* shown in Fig. 26. The slight distortion of separation between signal crossers, and related signal path deviations are the sole consequence of replacing ideal signal crossers with the 4×5 *cc*. The organ is larger, and its operation is slower but, the function is the same. Similar results are obtained for the tape construction control, the *halt gate*, the state machine, and the read/write head. More significant configuration change is confined to other organs.

The *tape mark* discriminator and associated packet signal squelch are shown in Fig. 35. Packet signal is sampled, and processed by a logical operator. Squelch occurs by side effect of construction; the now well examined and timely annihi-

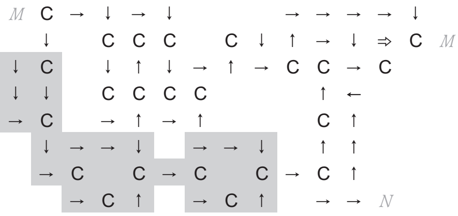


Fig. 35. The *tape mark* signal discriminator of the vNCA packet corpus. The shaded region identifies the *tape mark* decoder; it is a *logical and* operator over five variables.

lation and reconstruction of a confluent cell. We see here that packet signal can be engaged in complex function without encumbering crossing.

The memory unit for the vNCA packet corpus holds four bits within its register, and is composed of four one-bit memory cell assemblies, shown in Fig. 36. Registers of arbitrary size may be obtained by the stacking of these assembly units. The arrangement of assembly cells allows packet signal output along a path that is external to other corpus cells, and so never occasions necessity for signal crossing between memory unit operation and construction arm. The memory unit occupies a space of 1600 cells, comparable in size with the read/write head. More space efficient mechanisms which provide these functional benefits may exist. Not shown in the figure is that configuration which synchronises output from the memory cells, nor additional signal crossing organs. As part of the synchronisation process, the four bits taken from tape are prefixed by a logic high valued bit; packet signal is five bits long.

The vNCA packet corpus tape code is divided into two parts, as shown in Fig. 37, with low order bit patterns mapping to the corresponding logical inverse of construction signal, and high order patterns mapping to meta-instructions. The leading bit serves to anchor the bits of tape code, much like the decimal point anchors the digits of a real number. This allows the well ordered decoders of the signal discriminator to unambiguously distinguish all signal, save **OR** and **OU**, which are served last. Service of **OR** appears as an override to the **OU** service. Any signal not squelched by the signal discriminator is directed to the signal inverter; squelched signals are **TM**, **XU**, **XR**, **RL**, and **RD**. For signals **OU** and **OR**, packet signal is carefully altered before being passed to the signal inverter. The five-bit output of the signal inverter is selected by the synchronisation pulse emitted by the memory unit.

Input to the signal inverter occurs for the underlined codes shown in Fig. 37. At the terminus of the signal inverter is located a confluent state configured as a *logical and* operator. The operator is gated by the five bit signal `11111`, which is generated from the memory synchronisation pulse. The generated signal masks all the bits for the inverted packet signal, save the signals **OR** and **OU**. For these two signals, substitute packet signal is delivered to the inverter, yielding construction signal at output. By the mechanism of signal inversion, the entirety of those signal lines, the construction signal pulsers, and supporting signal crossing organs found in the construction arm control of the two pulse corpus self-replicators are subsumed. This is a significant reduction in configuration complexity.

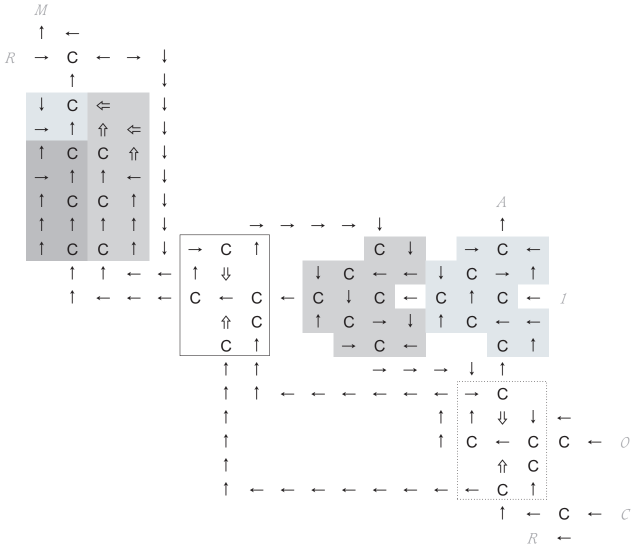


Fig. 36. A one-bit memory cell assembly for the vNCA packet corpus. The component one-bit memory cell is shown in three shades of grey in the figure upper left, and is a space optimised version of the one-bit memory cell given by Nobili and Pesavento [14]. The dark grey cells generate the signal stored in the light grey cells. The medium grey cells are the reset signal generator. The two boxed regions are cell-select gates, which indicate the cell of an assembly that shall receive data to store, according to the presence of the central left-pointing ordinary transmission state. Upon receipt of signal **0** or **1**, the cell-select gates are reset to closed; the next assembly in a register receives this signal via *A*, opening its cell-select gates. The clear signal is *C*.

We do not claim that the vNCA packet corpus is the smallest possible vNSRCA; we can envision means to reduce the size of the configuration presented. The greatest opportunity for reduction of configuration size comes with the memory unit, and configurations which implement long term signal delays. For instance, it may be possible that some repositioning of read/write head components will allow reduction in associated signal delay. Other organs seem unlikely candidates for configuration size reduction. We do not expect that any such configuration size reduction will be significant. Rather, it is only with complete removal of a major organ that significant size reduction may be obtained. Thus, we argue the vNCA packet corpus is, in terms of configuration size, a near optimal self-replicator. The endo-perimetric cell count for the vNCA packet corpus is 11,932. The tape is approximately 12 times the size of the configuration, giving a combined corpus and tape total of approximately 155K cells.

0000	CN	1000	sd	10000	11000
0001	SD	1001	XR	10001	11001
0010	SL	1010	od	10010	11010
0011	SU	1011	RD	10011	11011
0100	SR	1100	OU	10100	11100
0101	OD	1101	XU	10101	11101
0110	OL	1110	RL	10110	11110
0111	OR	1111	TM	10111	11111

Fig. 37. The instruction code of the vNCA packet corpus. Capitalised signal names correspond with expected codes; those in lower case are supported. Tape codes are shown at left; the prefixed form is shown at right. Underlined signal is served by the inverter.

We have thus demonstrated the correspondence between the cellular automata of Nobili and von Neumann, given self-replicators for both environments, and shown that the crossing of packet signal is not a requirement for the construction of self-replicators within either system of cellular automata. None of these self-replicators is a universal constructor. Our analysis of the mechanism of construction and the problem of signal crossing within self-replicating cellular automata suggests that the constructor which von Neumann was designing at the time of his death is, in spite of its self-replicability, rather less than universal. Indeed, given inclusion of active configuration within the product of construction, we hold that no constructor is universal, that for every constructor, one may find configuration which that constructor cannot construct, but which another, higher-order constructor can construct.

8 Modes of self-replication

Within cellular automata literature, we note a characteristic common to all given self-replicators, that their construction is completely at the direction of their mother. It may perhaps seem reasonable, even obvious, that a self-replicator should need all its parts before it may properly function. Indeed, we know of no published self-replicating cellular automaton which is able to function properly with even one cell of its configuration improperly configured, such as by being in one state when it ought be in another state. This is holistic self-replication.

Construction requires that a tape be read, that instructions be interpreted, and that the constructor know when to stop constructing. Self-replication requires that the construct described on tape be constructed, that the tape be replicated, that the read/write head should rewind along the tape so that it may be read twice, among other tasks. Yet, when constructing, the constructor need not know how to rewind the read/write head, nor how to replicate the tape, nor even how to stop reading the tape. Additional configuration needed for the performance of these tasks can be learned from the tape, and this just prior to necessity, if that configuration can be added without adversely affecting behavior [10]. This is self-replication by partial construction, a mechanism formally presented in Buckley [2].

We know of only one configuration capable of partial construction, an extension of the vNSRCA mentioned in Mange [9, pp. 1932]. Though we have not room here to present a thorough examination of the configuration, we mention a few important details. Implemented for vNCA, the configuration is composed entirely of pulsers, decoders and delays, and the vast majority of these organised into general recognisers, as described by Thatcher [16, pp. 149-150]. The only exceptional sub-configurations are the arm of the read/write head and the construction arm.

At the top level, the recognisers are organised into four coded channels, these layered one after the other, with a feedback relationship between some layers that is used to drive the read/write head. These coded channels implement a language translation scheme without the need for sub-configurations like memory units and signal squelches. Indeed, the only identifiable function of the standard architecture represented in this configuration is the supervisor. The ability of this automaton to perform any operation is therefore tied to the prior construction of recognisers that translate associated instruction, as represented in the form of signal.

The process of partial construction is that the mother configuration replicates the tape, and constructs a small but critical portion of the daughter configuration, which is then started. At this point, the mother retracts from the daughter, and the daughter is left to complete its own development. The daughter then reads its tape, following the instructions to complete construction of its configuration. The configuration arm must have access to all the functional components of the configuration, that they may be extended by new configuration. Further, all configuration must remain functional, even as new configuration is added. The configuration has but one construction arm, it alone adding configuration to the self-replicator; no *csc* is employed.

The vNSRCA mentioned in Mange has the interesting property that at no point within its configuration do signals actually cross; nowhere may be found an intersection between signal paths. Rather, signal paths diverge and converge, only. This configuration demonstrates that one may construct self-replicators without need for the crossing of signal within vNCA.

We assert that partial construction offers a rudimentary computational model of epigenetic developmental processes within self-replicating automata, and by extension, a mechanism for ontogeny within artificial life, and machines.

9 Results

We have reviewed the nature of signals and their crossing within von Neumann and Nobili cellular automata, derived and characterised specific signal crossing configurations, and applied these solutions to four self-replicators. We have demonstrated that pulse signal is sufficient to self-replication, and that packet signal can be serviced without crossing in self-replicators. We have given the smallest possible pulse signal crosser for vNCA, and given self-replicators that are markedly smaller than the design contemplated by von Neumann. We have included in the discussion relevant historical context of past efforts at solving the signal crossing problem. We have also reviewed the nature of configuration constructibility, and effective means of construction, including a conjecture respecting self-replication within vNCA without use of a taped description. Finally, we have presented the variation of self-replication by means of partial construction.

10 Comments

The historical content of our exposition is necessitated by our purpose, which is not directly the presentation of self-replicators. The ancillary provision of measurements of concrete example self-replicators, however served by this paper, is also not our purpose. Rather, our purpose is the comparison of signal crossing mechanisms as they affect the character of self-replicators, for both NCA and vNCA. The importance of historical context centers upon the performance of these solutions as applied within self-replicators, versus the justification of period researchers for ending their efforts to construct and demonstrate a vNCA self-replicator. Published literature records that perfect signal crossing was *judged* by period researchers as necessary. Specifically, Burks [4] offers,

> *"The von Neumann Memory Control required a real-time crossing organ (arrangement of parts) in order to implement the self-replicating encoded message repeatedly."*

Other fine examples are seen in the work of Lee [7, 8]. Instead, we have shown that one may efficiently implement a vNCA self-replicator, serving either pulse or packet signal, without need for perfect crossing. Indeed, and though we have not room here to present the result, a self-replicating vNCA configuration exists for which signal crossing is not required, even as it serves packet signal; Mange, et al. [9] mention this partial constructor.

Reiterating the importance of partial construction, we extend to readers the challenge given reviewers, a challenge which remains unsatisfied. Can the reader identify any self-replicating cellular automaton for which any ten cells of the initial configuration of said self-replicating cellular automaton (exclusive of any external tape description) may be changed from their non-ground state, into the ground state, such that the configuration retains the ability to self-replicate? We claim that only one such human artifact exists; our partial constructor.

Reviewers have suggested that referential use of private correspondence *is not helpful* to discourse such as that presented herein. We argue for the contrary case, and suggest that such correspondence acts to convert what may seem to be opinion or speculation into corroborated fact. Further, that such reference is to one yet living certainly implies the third party verifiability of supported statements; e.g. both Arthur W. Burks and Renato Nobili are easily contacted by email, and their email addresses are publicly displayed upon the internet. Those who question the factualness of our statements regarding correspondence with others are invited to do their own research.

We have mentioned that the Pesavento [15] design is not a self-replicator, and attribute this to a failure to construct one specific cell of its configuration. Nor will the Pesavento [15] design return to its initial condition, suitable to restarting; it is a one-time constructor at best. To this we should like to add that neither is the Pesavento [15] design an answer to the signal crossing problem of von Neumann, even if it did properly self-replicate. The NCA work of Pesavento [15] is an analog of the von Neumann problem, not a solution, and its acceptance as the latter by the scientific, and particularly the alife, community is an insult to von Neumann, and to those who value truth. That this fictional status continues to grace the Pesavento [15] design, especially by consequence of editorial inaction on the part of the corresponding journal, is unconscionable. This is our unequivocal opinion.

Readers may obtain full images of the four self-replicators presented in this paper in PDF form by download from the world wide web, at the URL `http://uncomp.uwe.ac.uk/automata2008/buckley/buckley.pdf`

11 Acknowledgments

We are deeply indebted to Arthur W. Burks and Alice Rowe Burks for their kind assistance with the historical context of prior work respecting signal crossing, and also for our discussion of the triple-return counter, which is a highly condensed restatement of Burks' own description [17, pp. 179-180].

Special thanks go to Hiroki Sayama for very helpful discussions.

Development of the configurations presented in this paper was greatly assisted by simulation software provided by Renato Nobili, department of physics, University of Padova, available gratis, by download from the world wide web at the URL `http://www.pd.infn.it/~rnobili/`

The state machine organs given herein, and within the catalog, and the memory organ given in the Nobili packet corpus, are derived from the corresponding organs given by Pesavento [15], and have been simplified and generalised.

Many thanks are due Ed Eckert, historian and archivist for Alcatel-Lucent Bell Labs, and also to Peter A. Benedict of Communications and Media Relations with Alcatel-Lucent Bell Labs, for their kind and able assistance in finding the original paper of Chester Lee [7] from within the Bell Labs Archives.

We thank the efforts of Andrew Adamatzky to bring this paper to publication.

The four holistic self-replicators discussed in this paper are available for demonstration by direct request of the author.

References

[1] Buckley, W. R. and Mukherjee, A.: Constructibility of Signal-Crossing Solutions in von Neumann 29-State Cellular Automata. In V. S. Sunderam, G. D. van Albada, P. M. A. Sloot, and J. J. Dongarra, (eds.) Computational Science - ICCS 2005. 5th International Conference on Computational Science (ICCS-2005). LNCS, vol. 3515, pp. 395-403. Springer-Verlag, Berlin (2005)

[2] Buckley, W. R.: Computational Ontogeny. Biological Theory: Accepted.

[3] Burks, A. W.: Von Neumann's Self-Reproducing Automata. In Burks, A. W., (ed.) Essays on Cellular Automata, pp. 3–64. University of Illinois Press, Urbana and London (1970)
Available from the DeepBlue server of the University of Michigan, on the internet at the persistent URL (URI) `http://hdl.handle.net/2027.42/3954`

[4] Burks, A. W. and Burks, A. R.: Private email correspondence dated January 24, 2008

[5] Goldstine, H. H.: The Computer from Pascal to von Neumann, Princeton University Press (1972)

[6] Hedetniemi, S. T.: Studies in Cellular Automata. Interim Engineering Report - Research in Theory of Adaptive Systems. ORA Project 06114 (1965)
Available from the DeepBlue server of the University of Michigan, on the internet at the persistent URL (URI) `http://hdl.handle.net/2027.42/4897`

[7] Lee, C. Y.: Synthesis of a Cellular Computer Using the 29-State Model of von Neumann. Bell Labs technical memorandum TM63-3344-5 (1963)

[8] Lee, C. Y.: Synthesis of a Cellular Computer. In Tou, Julius T. (ed.) Applied Automata Theory, pp. 217–234. Academic Press, New York and London (1968)

[9] Mange, D., Stauffer, A., Peparolo, L., and Tempesti, G.: A Macroscopic View of Self-replication. IEEE Proceedings, vol. 92, no. 12, pp. 1929–1945 (2004)

[10] McMullin, B.: John von Neumann and the Evolutionary Growth of Complexity: Looking Backward, Looking Forward... Artificial Life 6, pp. 347–361 (2000)

[11] Morita, K., and Imai, K.: A Simple Self-Reproducing Cellular Automaton with Shape-Encoding Mechanism. In Langton, C. G. and Shimohara, K. (eds.) Artificial Life V, pp. 489-496. MIT Press, Boston (1997)

[12] Mukhopadhyay, A.: Representation of Events in the von Neumann Cellular Model. J. of the ACM, vol. 15, no. 4, pp. 693–705 (1968)

[13] Nobili, R.: Private email correspondence dated December 3, 2003

[14] Nobili, R. and Pesavento, U.: Generalised von Neumann's Automata I: a Revisitation. In E.Besussi and A.Cecchini (eds.) Artificial Worlds and Urban Studies. DAEST, Venice (1996)

[15] Pesavento, U.: An Implementation of von Neumann's Self-Reproducing Machine. Artificial Life 2, pp. 337–354 (1995)

[16] Thatcher, J. W.: Universality in the von Neumann Cellular Model. In Burks, A. W., (ed.) Essays on Cellular Automata, pp. 132–186 University of Illinois Press, Urbana and London.
Available from the DeepBlue server of the University of Michigan, on the internet at the persistent URL (URI) `http://hdl.handle.net/2027.42/7923`

[17] von Neumann, J.: Theory of Self-Reproducing Automata. University of Illinois Press, Urbana and London. Edited and completed by A. W. Burks (1966)

Cellular automata and non-static image processing for embodied robot systems on a massively parallel processor array

Martin Hülse[1], David R. W. Barr[2], Piotr Dudek[2]

[1] Aberystwyth University, UK
msh@aber.ac.uk
[2] University of Manchester, UK
d.barr@postgrad.manchester.ac.uk
p.dudek@manchester.ac.uk

Abstract. A massively parallel processor array which combines image sensing and processing is utilized for the implementation of a simple cellular automata (CA). This CA is essential part of an image processing task supporting object detection in real-time for an autonomous robot system. Experiments are presented, which demonstrate that objects will be detected only if they move below a specific velocity in the visual scene. Based on these experiments we will discuss the role of configurations changes if a CA is seen as a parallel processing computer. This leads us to the conclusion that if CA are performing non-static data processing tasks they might be better approached as sensor-driven parameterized dynamical system rather than as parallel computers operating on initial configurations only.

1 Motivation

Recent progress in chip design provides massively parallel processor arrays where each processor is linked to its neighbor processors. Such systems are ideally suited to perform low-level pixel-parallel image processing tasks. The SCAMP (SIMD Current-mode Analogue Matrix Processor) vision chip is one example of such a parallel processor array that integrates image sensing and processing on a single silicon die providing low-cost, low-power and high-performance vision system for autonomous robot systems [5].

Unfortunately, the majority of current low-level pixel-based image processing approaches are focused on 2-dimensional convolution operators and the analysis of static features in static images. However, image processing in autonomous robots is rather confronted with permanently changing signals and noise.

On the other hand, architecture and data processing of the SCAMP vision system are able to instantiate 2-D cellular automata (CA). It is well known that CA provide a rich reservoir of dynamical properties and behavior on a global scale [7]. Our assumption is that nonlinear dynamical properties generated by CA are a promising substrate for the processing of complex and changing image data for autonomous robot systems. However, methods and frameworks are missing which allow a targeted design of CA supporting specific processing tasks on image data. The following investigation is a first step towards CA-based processes with high update rates (100 kHz) performing image processing in real-time in embodied autonomous robot systems.

This contribution summarizes first simple experiments that demonstrate how a CA can support image processing for a robot platform in a changing environment (including an active vision system and a manipulator). In particular we will present an image processing task performed on different time scales. Based on this example we will discuss the utilization of CA as data processing devices. This discussion emphasizes the importance of changes in CA configurations during the evolution of the system (here caused by visual input data). This view might be contrary to traditional approaches, where CA are seen as discrete dynamical system working as a parallel processing computer, "where data is considered to be the initial CA configuration" [7]. Our hypothesis is that an application of CA for "non-static" / robotic related image processing has to consider CA as sensor-driven parameterized dynamical systems [6]. In consequence, the data the system is operating on are the parameter values of the CA and their changes over time caused by the visual input.

2 The SCAMP vision system

The SCAMP vision chip is a sensor/processor device, including a 128x128 array of processor cells, arranged in a 4-connected neighborhood. Each processor also integrates image sensor, for pixel-parallel image input. The processors operate in a single-instruction multiple-data (SIMD) mode, i.e. all processors execute the same instruction, issued by a central controller. The instruction set includes arithmetic operations and neighbor transfers, which allows implementation of a variety of pixel-parallel local computations, characteristic of early vision applications. SCAMP executes instructions at 1MHz rate, so that even fairly complex operations can be easily implemented at video frame rates.

The general-purpose, software-programmable nature of the device enables the execution of a variety of algorithms. In particular, CA can be easily implemented, by writing simple programs to execute state update rules. The neighborhood needs not be restricted to 4-connectivity, as data can be easily passed between processors in several steps over larger distances.

The processors in the SCAMP chip operate on analogue data values, i.e. unlike a conventional digital processor it natively operates on real-valued numbers, albeit with a limited signal/noise performance and constrained dynamic range. The out-of-range operation results in a soft saturation (sigmoid function), which can be exploited to implement nonlinearities in the system [2]. Logic operations, and discrete state-space CA can be implemented using thresholds (comparison operations), however the native mode of operation of the chip is continuous-value.

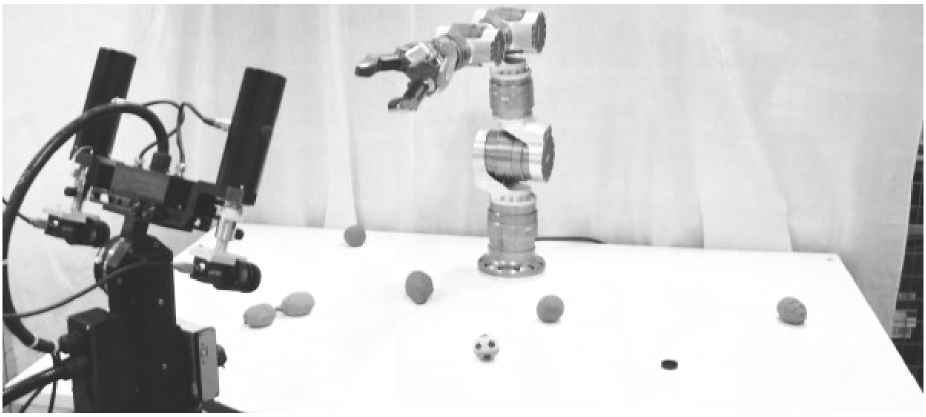


Fig. 1. Robotic arm and manipulator system mounted on a table and the active vision system, scanning the scenario with two cameras and a SCAMP vision system.

3 The use of SCAMP in a robotic setup

The robotic scenario in which SCAMP is used consists of an active vision system and a robot manipulator with 7 degrees of freedom (DOF). As one can see in Fig. 1, the robot manipulator is mounted on a table. Sensory input driving reaching and grasping is delivered from the active-vision system scanning the table. Apart from SCAMP two additional cameras deliver visual inputs that can drive the pan, tilt and verge motors of the active vision system. In the following we present experiments where only SCAMP is used for visual input and the pan-tilt-verge configuration is fixed. In other words the SCAMP camera doesn't move while scanning the table.

Confronted with such a robot system the first step is the implementation of control systems performing goal-directed and robust gaze and reaching behavior. However, for autonomous robots it is important that the system itself, i.e. without human interaction, is able to trigger action sequences for reaching or gazing. In consequence, the system needs to measure not only where an object is located on the table but also whether or not it is moving, and if so, does it move slow enough in order to reach it with its manipulator.

Due to the physical reality of our robot hardware, a simple pan-tilt-system, obviously, can operate in a much faster way than an 7 DOF robot arm. Successful reach movements can only be performed with object much slower than it is necessary for a visual based object tracking with our pan-tilt system. Therefore, we have implemented a process that detects objects in real-time depending on the speed the objects move in the visual field. If an object is faster it becomes invisible to the system. If an object emerges in the visual field then it is implicitly given that it doesn't move at all or at least slow enough to reach for it. The

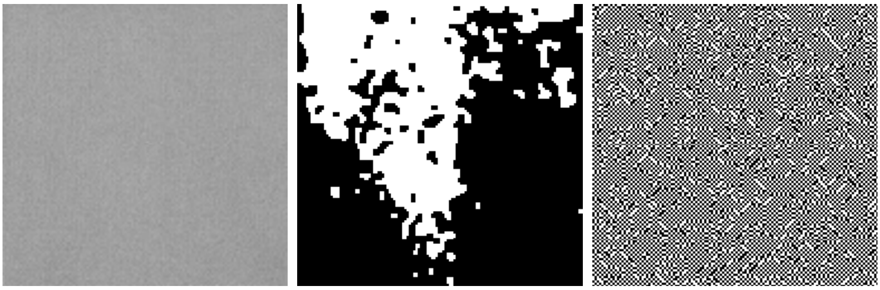


Fig. 2. Resulting processor values staring from an random initial configuration (a). The grey values representing values around zero. The patterns in (b) emerge from the inhibitory coupling ($w = -0.25$), while the pattern in (c) is generated by the excitatory linking ($w = 0.25$).

sensitivity to changes in the visual input depends on a single parameter and therefore the system can easily be adapted to different time scales of the different actuators in the robot system (here we have a manipulator and a pan-tilt-verge system). The core application of this image data processing task is provided by a CA. The CA is instantiated by the SCAMP system, which we will explain in the following section.

4 Embedding cellular automata into a robotic scenario

The structure of the applied CA is very simple and straightforward to implement with SCAMP. Each processor P_n receives input from its four direct neighbor processors $P_{n,N}$, $P_{n,S}$, $P_{n,E}$ and $P_{n,W}$. Due to the analogue character of SCAMP operating on currents, we have continuous values for all P_n. As we have already mentioned, the lower and upper saturation domain resulting from out-of-range operations match with a sigmoid function f [2]. The system can be formally written as:

$$P_n(t+1) = f\left(w \cdot \left(P_{n,N}(t) + P_{n,S}(t) + P_{n,E}(t) + P_{n,W}(t)\right)\right),$$

where $w < 0$ (here $w = -0.25$). In fact this CA implementation on SCAMP is continuous-value CA with a nonlinear update rule, or as it is called in the literature a discrete-time cellular neural network (DT-CNN) [4].

Since the values of the processors can be positive or negative this inhibitory linking scheme generates an interplay of excitation and inhibition. Without visual input and random initialization, the processor array evolves to a state where the value of a processor is very likely the same of its direct and indirect neighbors. This is represented by large regions and patches colored either black or white, as shown in Fig. 2(b). These black and withe regions indicate the emergence of clusters and closed regions containing processors whose values are driven into the

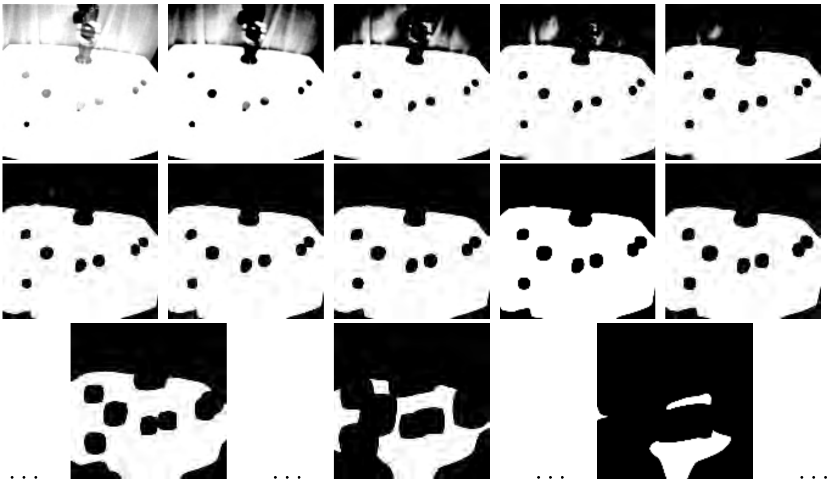


Fig. 3. Sequence of CA configuration (time step: 0,. . . , 9, 50, 70 and 90) without resetting and continuous image data input.

same saturation domain, either in the upper or lower saturation. It is interesting to see, that the opposite is the case for positive links between the processor (i.e. $w > 0$). Such an excitatory coupling generates almost a chess-board pattern, i.e. each direct neighbor processor is driven into the opposite saturation domain, Fig. 2(c).

Considering the negative coupling between the processors it seems that the growing black patches are perfect to indicate salience region within an image. Therefore, we implemented the CA with negative couplings on SCAMP and further on, used the visual data provided by SCAMP (128x128 grey value image) as permanent input for this specific CA. Hence, the CA configuration at time step $(t + 1)$ is determined by the configuration at time t and the current value of the corresponding pixel in the image $(PIX_n(t))$:

$$P_n(t+1) = f\left(PIX_n(t) + w \cdot (P_{n,N}(t) + P_{n,S}(t) + P_{n,E}(t) + P_{n,W}(t))\right),$$

where $w = -0.25$. In Fig. 3 an example of the resulting sequence of images is shown. The black circular regions in the image which represent the objects on the table are growing, even if initially they have very low grey values. This is the result of adding the current visual input in each time step.

As the sequence in Fig. 3 is indicating, such a setup seems rather inapplicable for non-static image data, i.e. moving objects or a moving camera. After a number of time steps the whole image is black, because the black regions never disappear again and the system becomes “blind”. In order to stay sensitive to new visual stimuli we setup an additional reset mechanism, which resets the CA configuration every n time steps. The CA configuration before the reset is the output of our image processing. In this way the image processing is actually

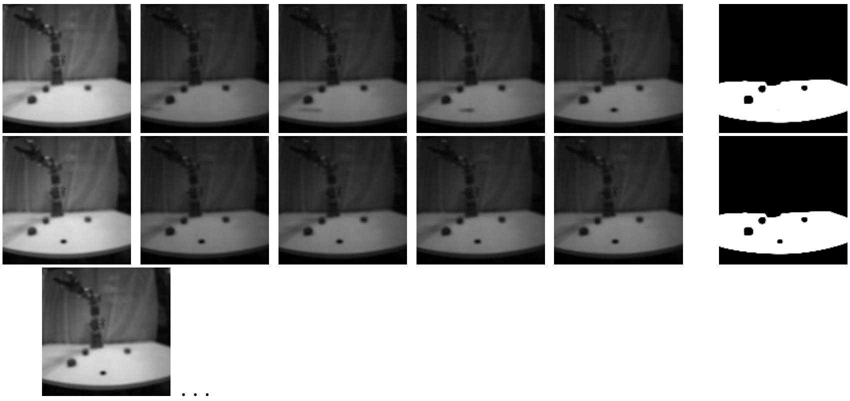


Fig. 4. Sequence of images representing the visual input PIX and the CA configuration between the resets, done every 6th time step. The six images in each line represent one period starting with the first state after the reset and ending with the last state before reset. The first five images in each line represent the visual input PIX at the first five time steps after reset. The last image represents the CA configuration before it is reset, i.e. the actual output data of the process

established by two processes: the inner loop that updates the CA involving the current visual input of SCAMP, and the outer loop reading the CA configuration every n steps and resetting the CA configuration. As we will show in the next section the output stream generated by the outer loop provides the properties we are aiming for: objects on the table only emerge in the output image, if their speed in the visual image is below a certain value.

5 Experiments with moving objects

In the following experiments we have reset the system every six time steps ($n = 6$). In order to give the reader an impression of the relation between output and visual scene, we have plotted six images between each reset. The first five images in this sequence represent the visual input data PIX for the first five steps after each reset. The last image in this line the actual output / result of the process. It is the CA configuration its reset.

In the first example shown in Fig. 4 we see at the front part of the table a small object sliding into the scene. The first five images in the first line show this event. The faster the object moves the more blurred it appears in these images. Notice, these images represent the visual input data PIX which are fed into the CA. The last image in this line shows the CA configuration representing the actual output of this process. One can see, that the new object doesn't emerge in this image, despite the fact that it is clearly visible in four out of five PIX images. Once the object has stopped on the table (second line of images) it

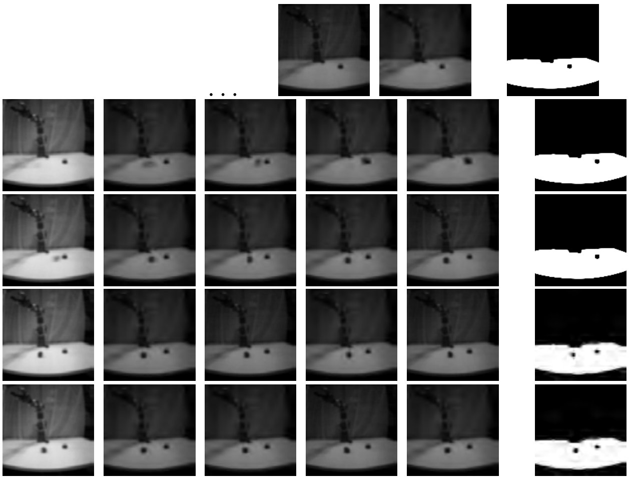


Fig. 5. The representation of image data processing as in Fig. 4. See text for explanation.

emerges in the output image. In other words, only after it has stopped it becomes visible for the robot system.

Another example of object detection is shown in Fig. 5. There the new object remains invisible for more than 12 time steps because it is moving to fast for this configuration.

6 Conclusion

We have shown that a CA can provide image processing for an autonomous robot system. Objects are only detected by the vision system if they move below a certain speed. This quality of object detection simplifies the triggering of reaching actions because if a stimulus emerges then it is guaranteed that it isn't moving to fast for the arm system and the robot can directly perform the related action sequence without any further data processing.

The "insensitivity" to fast moving objects is determined by the frame rate SCAMP is providing visual input. Within certain boundaries this frame rate is a free parameter of the SCAMP vision system and therefore, this process can easily be tuned to different time scales, even online.

The crucial element for detecting slow moving objects only is the continuous input of new visual data into the CA at each time step. As the image sequences in Fig. 4 and 5 clearly show occasional and limited appearance of low activities in the visual data (low grey values) doesn't immediately drive the output values into the lower saturation. Only if a low activation in a region of high activations is continuously measured then the corresponding CA units change and indicate a object in the visual scene. This mechanism can be seen as accumulation of evidence and is frequently discussed for action-selection processes in biological and artificial systems [3].

However, CA are usually referred to as computational devices that operate only on initial configurations. The setup here could be seen as a counterpart of this approach and we believe, our experiments give evidence that it is worth investigating CA operating with continuously changing inputs. Nevertheless, there is a fundamental difference between a CA operating on initial configurations only and a CA driven by permanent changing inputs. With respect to the update rule describe above we can describe three principal ways CA can operate on image data. The first possibility is that image data are only used for determining the initial state of the CA, i.e. each processor P_n is initialized with the value of the corresponding pixel in the image:

$$P_n(0) = PIX_n(0).$$

After this initialization the system is updated according to the update rule which doesn't involve any image data:

$$P_n(t+1) = f\left(w \cdot (P_{n,N}(t) + P_{n,S}(t) + P_{n,E}(t) + P_{n,W}(t))\right).$$

In fact we have a dynamical system without free parameter. Hence, its behavior is determined by its initialization, a single image, only.

Another way of combining visual data and CA uses the image data for the definition of a constant offset:

$$O_n = PIX_n(0)$$

for each unit in the CA. This offset is a parameter that now determines the update rule as follows:

$$P_n(t+1) = f\left(O_n + w \cdot (P_{n,N}(t) + P_{n,S}(t) + P_{n,E}(t) + P_{n,W}(t))\right),$$

where $P_n(0)$ is arbitrary chosen. In this case we have a parameterized dynamical system [1]. The system behavior of the CA is determined by the initialization *and* the parameter O_n for each processor.

Comparing these two cases with the original update rule applied for our robotic experiments:

$$P(t+1) = f\left(PIX_n(t) + w \cdot (P_N(t) + P_S(t) + P_E(t) + P_W(t))\right).$$

we see that our setup is the only process that can deal with changing image data. In contrast to the other systems, where there is no way of feeding data changes into the running process.

To sum up, the here introduced CA is, precisely speaking, an implementation of a parameterized dynamical system, whose parameters change according to the image data. This is, what we call a sensor-driven parameterized dynamical system, because parameter changes are driven by the image data which is the sensory input of an autonomous robot system. Sensor-driven parameterized dynamical systems seem to be a promising approach to deal with changing inputs, since the two other alternatives of combining CA and image data are only operating on static data. This observation let us conclude that CA should be approached as sensor-driven parameterized dynamical systems, if one is interested in applications and novel computational paradigms for embodied robot systems.

Further on, as we have outlined above our continuous-value CA can also be seen as discrete-time cellular neural networks. Research on DT-CNN has already emphasized and exemplified the need of continuous input for visual data processing [4]. However, as it is often the case in the domain of complex systems, the phenomena are well know, but the challenge is to develop related applications as well as to derive general mechanisms that might lead to alternative paradigms for information processing. In this paper we have demonstrated that the SCAMP system is able to create complex dynamics based on CA in real-time. Therefore, embodied robot systems equipped with a SCAMP vision chip seem to be an efficient framework for bringing together the complex dynamics of CA *and* complex image processing in real-time. In addition, autonomous robot systems together with the application of self-organized adaptation processes (e.g. evolutionary algorithms) provide a context where the use of complex dynamics for real-world scenarios can be systematically explored and analyzed. This approach was demonstrated successfully within an evolutionary robotics framework for small discrete-time dynamical systems (small with respect to the state space) [6]. We believe SCAMP is a promising tool for scaling up this approach massively.

Acknowledgement: This research has been supported by the EPSRC through grant EP/C516303/1.

References

[1] V. I. Arnold, *Geometrical methods in the theory of ordinary differential equations*, (Springer,1983).

[2] D. R. W. Barr, P. Dudek, J. Chambers and K. Gurney, “Implementation of Multilayer Leaky Integrator Networks on a Cellular Processor Array” (in International Joint Conference on Neural Networks, IJCNN 2007, Orlando, Florida, 2007).

[3] R. Bogacz, K. Gurney, “The basal ganglia and cortex implement optimal decision making between alternative actions” (in Neural Computation, 19, 442-477, 2007).

[4] L. O. Chua, T. Roska, *Cellular neural networks and visual computing - Foundations and applications*, (Cambridge University Press, 2001)

[5] P. Dudek, S.J.Carey, “A General-Purpose 128x128 SIMD Processor Array with Integrated Image Sensor” (in Electronics Letters, 42(12), 678-679, 2006).

[6] M. Hülse, S. Wischmann, P. Manoonpong, A.v. Twickel, F. Pasemann, “Dynamical Systems in the sensorimotor loop: On the interrelation between internal and external mechanisms of evolved robot behavior” (in M. Lungarella et al. (Eds.) 50 Years of Artificial Intelligence, LNCS 4850, Springer, 186-195, 2007).

[7] A. Wuensche, M. Lesser, *The global dynamics of Cellular Automata* (Addison Wesley, 1992).

Evolving one-dimensional radius-2 cellular automata rules for the synchronization task

Alexsandro Souza Mariano and Gina Maira B. de Oliveira

Universidade Federal de Uberlândia, Artificial Intelligence Laboratory
Uberlândia MG, Brazil
gina@facom.ufu.br

Abstract. Although being very simple to implement cellular automata (CA) are able to perform complex computations. The understanding of how they perform these computations is still vague. An efficient approach is the use of evolutionary algorithms, as the genetic algorithm (GA), to search for CA able to exhibit a desired computational behavior over their rule space. One of the most studied computational tasks in this context is the synchronization task (ST). In a previous work, a heuristic based on four forecast behavior parameters was incorporated to guide evolutionary search returning better radius-2 CA rules to solve ST. Here, we analyze the influence of incorporating a heuristic based only on two of these parameters into the genetic search. We also investigated a simple strategy to preserve the diversity of the population formed by radius-2 transition rules. As a result of these investigations nine radius-2 CA rules that solves ST with a good efficacy were found; all of them are better than the best one previously published.

1 Introduction

The dynamics of a cellular automaton is associated with its transition rule. In order to help forecast the CA dynamical behavior, several parameters have been proposed, directly calculated from their transition table [8, 2, 20, 13]. However, the decision problem associated with precisely forecasting the dynamic behavior of a generic cellular automaton, from arbitrary initial configurations, has been proved to be undecidable [3]). Thus, any set of parameters can only be expected to *help* forecast the dynamic behavior of a cellular automaton.

Cellular automata have the potential to act as abstract machines that can perform arbitrarily complex computations from local operations. Several researchers have been interested in the relationships between the generic dynamic behavior of a cellular automaton and its computational abilities as part of the more encompassing theme of the relationships between dynamical systems and computational theories [18, 19]. Various investigations have been carried out on the computational power of CA, with concentrated efforts in the study of one-dimensional CA capable of performing computational tasks [10]. One of the approaches in this kind of research is the use of evolutionary methods as search procedures to find CA with the predefined computational behavior [4, 9, 5, 12].

Here, we show results obtained in a computational task named Synchronization Task, ST for short, which was proposed in [4] and also investigated in [11, 17, 14]. In this task, the goal is to find a one-dimensional, binary cellular automaton that, given an arbitrary initial configuration, it has to reach a configuration that cycles through two lattice types: all cells in state 0 in one time step, and all cells in state 1 in the next time step.

Four forecast parameters have been used before as an auxiliary metric to guide the GA optimization searching for rules able to perform ST [14]. The best radius-2 CA rule for this task previously published presented an efficacy bellow 95% [14]. Here, we investigate the use of only two of them: sensitivity [2] and neighborhood dominance [13]. Nine radius-2 CA rules with efficacy above 95% were found; the best rules have efficacy around 96.3%.

2 Forecast behavior parameters

Through analysis of the dynamic behaviour exhibited by CA, it becomes clear they can be grouped into classes. A few rule space classification schemes have been used in the literature; for instance, Wolfram [18, 19] proposed a qualitative behaviour classification, which is widely known. Later on, Li and Packard [8] proposed a refinement to the original Wolfram classification, which divides the rule space into six classes: Null, Fixed Point, Two-Cycle, Periodic, Complex (or Edge of Chaos) and Chaotic.

The dynamics of a cellular automaton is associated with its transition rule. In order to help forecast CA dynamical behavior, several parameters have been proposed, directly calculated from its transition table [2, 7, 8, 13, 20]. CA rule spaces with high cardinality make their parameterization a daunting task, which entails the usage of more than a single parameter in order to achieve a better characterization [2, 13].

It is not possible to expect the precise forecasting of a generic cellular automaton, from an arbitrary initial configuration. It has been proved, in [3], that the decision problem associated with the latter generic proposition is undecidable. Hence, all we can expect is really a parameter that can help forecast the dynamic behavior of a cellular automaton; in particular.

A set of five forecast parameters were selected in [13] and four of them were applied in the ST experiments reported in [13]. In the evolutionary experiments described in Section 5, only two parameters from that set were used: sensitivity and neighbourhood dominance. Informally, they can be described as follows:

Sensitivity : is defined as the number of changes in the outputs of the transition rule, caused by changing the state of a cell of the neighbourhood, each cell at a time, over all possible neighbourhoods of the rule being considered [2].

Neighbourhood dominance : quantifies how much change is entailed by the CA rule transitions, in the state of the centre cell, in respect to the state that predominates in the neighbourhood as a whole. The parameter value comes from a weighed sum of the number of transitions of the CA rule in which neighbourhood dominance occurs, with the additional feature that, the more homogeneous the neighbourhood involved, the higher its weight [13].

3 Computational tasks

Cellular automata have the potential to embody models of complex systems, and to act as abstract machines that perform complex computations with high degree of efficiency and robustness. Various investigations have been carried out on the computational power of CA, with concentrated efforts in the study of one-dimensional CA capable of performing computational tasks [10]. The most widely studied CA task is the density classification task (DCT) [9]. In this task the objective is to find a binary one-dimensional cellular automaton that can classify the density of 1s in the initial configuration of the 1D lattice, such that: if the initial lattice has more 1s than 0s, the automaton should converge to a null configuration of 1s, after a transient period; otherwise, it should converge to a null configuration of 0s.

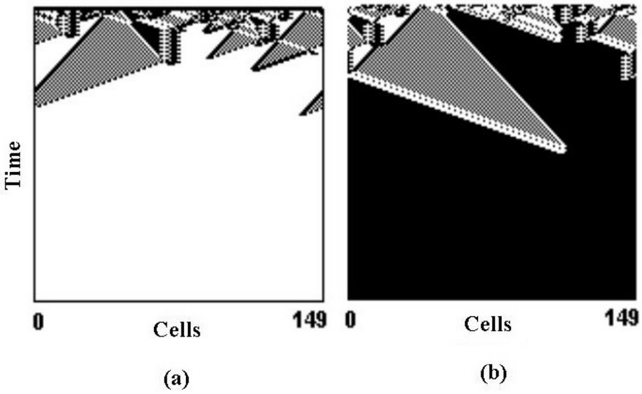


Fig. 1. Density Classification Task. Rule: 070447470700460705774757F777FF77

Figure 1 presents two space-time diagrams of a CA rule successfully performing DCT. It is a nontrivial task for a small-radius CA, since they rely only on local interactions. On the other hand, DCT is trivial for a system with a central controller or a central storage [10]. Performing this task well for a fixed lattice size requires more powerful computation than can be performed by a single cell or any linear combination of cells. Since the 1s can be distributed throughout the CA lattice, the CA must transfer information over large distances. A kind of global coordination is required to communicate cells that are separated by large distances and that cannot communicate directly. However, it has been proved that no finite-radius, two-state CA with periodic boundary conditions can perform this task perfectly across all lattice sizes [6].

Another computational task previously studied was the synchronization task, ST for short [4]. In this task, the goal is to find a one-dimensional, binary cellular automaton which, given an arbitrary initial configuration of a lattice of N cells, after T time steps, it should reach a configuration that cycles through two lattice types: all cells in state 0 in one time step, and all cells in state 1 in the next time step. T is the minimum time required to guarantee synchronization of all cells, a parameter that depends on the lattice size (N).

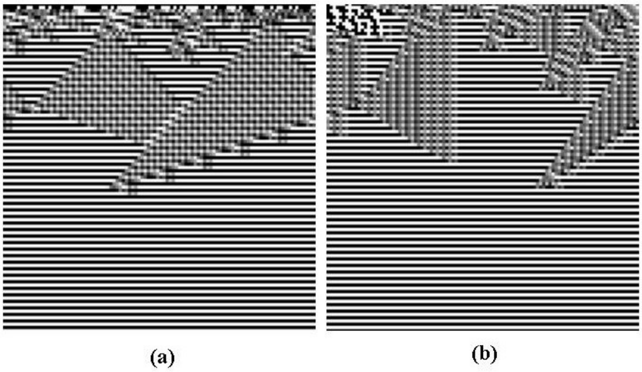


Fig. 2. Synchronization task: (a) Radius-3 rule: FEB1C6EAB8E0-C4DA6484A5AAF410C8A0 (b) Radius-2 rule: EAC38AE8

Figure 2 presents space-time diagrams of two different CA rules solving ST: the first is a radius-3 CA rule and the second is a radius-2 one. The task is also nontrivial since a small-radius CA employs only local interactions while synchronous oscillation is a global property. The locality of interaction can directly lead to regions of local synchrony and it is more difficult to design a CA that will guarantee that spatially distant regions are in phase [4]. An effective CA rule must transfer information over large space-time distances. There are several one-dimensional radius-3 CA rules able to solve ST for any arbitrary lattice configuration [4]. The radius-3 rule used in Figure 2 solves this task with 100% of efficacy. Considering radius-2 rules, it is known that there are good rules with efficacy above 90% [14] but the upper bound limit is unknown. Figure 2 shows the evolution of a radius-2 CA rule which solves ST with around 95% of efficacy [14].

4 Evolving cellular automata rules

Once a computational task is defined, it is not easy to find a cellular automaton rule that performs it. Manual programming is difficult and costly, and exhaustive search of the rule space becomes impossible, due to its high cardinality. A practical alternative has been the use of search and optimisation algorithms, particularly evolutionary computation methods [1, 4, 5, 11, 12, 14].

Packard [16] was the first to publish results using a genetic algorithm as a tool to find CA rules with a desirable computational behavior. He considered one-dimensional CA rules as individuals in a population and defined their fitness according to their ability to perform the specified task. In this way, the genotype of the automaton was given by its transition rule, and the phenotype by its ability to perform the required task. Crossover among two CA was defined by the creation of two new transition rules, out of segments of two other rules; mutation was achieved by the random flip of the output bit of one of the transitions of the rule.

Other evolutionary computation techniques were used to find such kind of CA rules. Genetic programming was also used as a search procedure for CA rules to perform the density classification task [1]. Furthermore, for the same task, the best known radius-3 rule was obtained by a coevolutionary approach [5]. It has an efficacy of 86% in performing DCT.

Evolutionary methods were also used in the synchronization task. ST was proposed by Das, Crutchfield, Mitchell and Hanson [4] and it was also investigated in [11, 14, 17]. In the experiments reported in [4] genetic search was used to look for radius-3 rules able to perform the synchronization task. Several radius-3 rules that perform the task with an efficacy of 100% were obtained. Although the authors actually found perfect radius-3 rules, they did not obtain success in CA with radius-2. In the latter space, they only found rules with virtually 0% of efficacy [4]. Later, Oliveira and collaborators [13] had evidenced that a simple GA as the employed in [4] was able to find radius-2 rules with efficacy around 80%. Moreover, by introducing a kind of parameter information it became possible to find radius-2 CA rules with more than 90% of efficacy solving ST.

In previous works, a simple GA environment was modified to incorporate a heuristic based on some forecast parameters and it was used to find DCT and ST rules [12, 14]. This approach uses parameter bands where good rules are more likely to occur. Once this information is available, it is used in an active way, as an auxiliary metric to guide the processes underlying the GA search.

In [12], four forecast parameters (sensitivity, neighborhood dominance, absolute activity and activity propagation) had been calculated for six radius-3 CA rules that performs TCD with good efficacy (above 80%) found in literature. The parameters values were normalized between 0 and 1. It was observed that, although the published rules were obtained by distinct search methods, the parameters fits in relatively narrow bands. The information of these bands was incorporated as an auxiliary metric of the evolutionary search. It was inserted through a heuristic that returns a value as bigger as closer are the rule parameters to these bands. The best rule found in 100 executions of this experiment has an efficacy of 80.4%. In another experiment performed without the incorporation of the heuristic, the best rule has an efficacy of 76.4%. In [15] the parameter-based heuristic was better investigated. An analysis of the individual contribution of each parameter in the improvement of the evolutionary search was made and it was evidenced that the parameters sensitivity and neighborhood dominance are the most important in DCT search.

In [14] a similar method was adapted for the synchronization task using CA rules with radius-2. A first experiment with a standard GA was executed without the information of the parameters. The best rule found achieves 82.8% of efficacy. From the analysis of the best rules found in this experiment, desirable bands related to the four parameters had been gotten: sensitivity (0.36 to 0.38), neighborhood dominance (0.17 to 0.18), absolute activity (0.69 to 0.72) and activity propagation (0.36 to 0.41). This information was incorporated and GA was executed again. More rules with good efficacy had been gotten in the parameter-guided experiment; the two top rules have efficacy of 94.2% and 93.2% (measured in 10.000 different lattices of 149 bits), being the first one the best published radius-2 CA rule for ST as far as we know. The hexadecimal code of this CA rule is EAC38AE8; a space-time diagram of this rule was presented in Figure 2.

5 Evolutionary environment

The individual contribution related to each parameter that composes the heuristic used to guide DCT in [12] was better investigated in [15]. One of the main conclusions is that sensitivity and neighborhood dominance were the most important parameters involved in this heuristic. Thus, a parameter-guided evolutionary environment was implemented in [15] which allows to use different combinations of parameters in this search. The better results were obtained with the couple of parameters cited before. Also, the results obtained with them in DCT were better than those obtained with the heuristic composed by four parameters in [12]. This good performance with DCT motivated us to develop a similar approach to ST search.

The conception of the new environment was strongly based on the GA described in [4] and in the parameter-guided evolutionary environment described in [14]. Each individual of the population represents a radius-2 transition rule and it is directly represented by a binary sequence of 32 bits. The environment evolves a population of 100 individuals over 50 generations. At each generation, each individual evaluation was obtained out of testing the efficacy of the automaton in synchronizing a sample of initial configurations (IC) and in the adequacy of its parameters to desirable pre-specified ranges. Additionally, elitism was used at a rate of 20% (that is, the 20 best rules of the population at each generation were always preserved to the next); the other 80 individuals were obtained through crossover and mutation. Parent selection for the crossover was made directly from the elite, without considering individual fitness. Standard one-point crossover was used at a rate of 80%. Mutation was applied after crossover, in each new generated individual, at a rate of 2% per bit.

The two-parameter-based heuristic was incorporate in GA environment in a similar way as done in [14], in which four parameters were used. Basically, the parameter-based heuristic is coded as a function (referred to as *Hp*), which returns a value between 0 and 100 for each transition rule, depending on the rule's parameter values.

	Sensitivity		Neighborhood Dominance	
	$Sens_{min}$	$Sens_{max}$	Dom_{min}	Dom_{max}
Range 1	0.35	0.49	0.15	0.19
Range 2	0.40	0.55	0.15	0.25
Range 3	0.42	0.53	0.17	0.21

Table 1. Desirable Parameter Ranges.

The first way to establish the heuristic is supplying the value ranges of the parameters in which it would be desirable to obtain the CA rules, for some *a priori* knowledge of the problem. Table 1 shows different ranges used in synchronization task experiments considering sensitivity ($Sens_{min}$, $Sens_{max}$) and neighborhood dominance ($Domi_{min}$, $Domi_{max}$). The ranges of each parameter in Tab. 1 are interpreted as those with high probability of occurrence of the desired behavior (that is, the ability to solve ST). In the case of the sensitivity parameter and considering the first desirable range in the table, for example, it indicates that there is a high probability of the rule being able to perform the ST if its sensitivity is between 0.25 and 0.75. *Hp* component related to sensitivity depends on its value, according to the following: if the parameter value is within the interval ($Sens_{min}$, $Sens_{max}$), the evaluation is 100 (maximum value); otherwise, the value linearly decays. A similar calculus is done for *Hp* component related to neighborhood dominance. Finally, *Hp* value is a single average of the two components. That is, the two parameters equally contribute to heuristic. *Hp* is then used to bias the GA operation, in the following ways:

Selection : the fitness function of a cellular automaton rule is defined by the weighted average of the original fitness function used in [4] — efficacy in synchronizing 100 random ICs — and the *Hp* function. The fitness function used for selection (F), is a composition of the two previous functions, the parameter-based component being weighted by W_h, and expressed as follows: $F = F_o + W_h * Hp$
Hence, if $W_h = 0$, the fitness function will be the same as that used in the original experiment [4].

Crossover : N_{CROSS} different crossovers are made, at random crossover points, creating twice as much offspring; the two of them with the highest *Hp* value are then considered for mutation.

Mutation : N_{MUT} distinct mutations are made to a rule, so that the one with the highest *Hp* value is selected for the new population.

In the parameter guided experiments reported in section 4, it was used W_h=0.4, N_{CROSS}= 10 and N_{MUT} =10. CA specification was the same as in [4, 14]:

- Time steps in CA evolution: 320.
- Lattice size: 149 bits.
- Number of initial configurations (IC) used in the population evaluation at each generation: 100 ICs. As explained in [4], these lattice configurations are uniformly sorted to improve GA convergence.
- Number of initial configurations (IC) used in the final evaluation at last generation: 10.000 ICs. These lattice configurations are randomly sorted. The top 10 rules of the final population are submitted to this evaluation which is more severe than the other one performed at each generation: the sample size is 100 times bigger and the randomly generated IC is the worst case to synchronize. The best one CA rule (out of the 10 rules tested) is returned as the final result of the GA run.

The environment previously described is very similar to the one implemented in [14] except to the number of parameters involved in heuristic *Hp*: two parameters instead of the four. However, after a series of experiments had been performed, which will be described in the next section, something was enhanced when GA dynamics was analyzed: there were a lot of repeated individuals in the final population. The majority of experiments conducted in evolving cellular automata rules are strongly based on the environment described by Mitchell, Hraber, and Crutchfield [9], which was developed to search radius-3 CA for DCT. In this cited environment there is no extra strategy to preserve the diversity in the rule population. By our own experience with DCT search and ST search, the mutation rate is enough to maintain a good diversity in the population when working with radius-3 rules which are formed by 128 bits. However, in the experiments described in this work the GA manipulates radius-2 CA trying to solve ST in this rule space. Such kind of rules is formed by 32 bits and apparently the mutation is not sufficient to maintain a desirable diversity. Thus, we inserted in our environment a simple strategy to avoid repeated individuals described as follows. When a new solution is created, through the application of crossover and mutation operators, before its insertion in the current population it is checked if an individual identical to this one already exists. If so, the new individual (the copy) is inserted with an evaluation equal to -1. It makes any individual that is not a copy of some other one better than any copy. Even if a copy survives to the next population, it has few chances to be selected, since the selection method is very elitist and uses only the 20% best individuals to form crossover pairs. Therefore, the chances of crossovers with different parents increase a lot. Without this strategy, we observe that several crossovers were made with identical individuals resulting in identical offspring.

6 Results

6.1 Experiments

The initial experiments were performed with an environment in which no strategy to preserve diversity was implemented. Different desirable parameter ranges were evaluated as heuristic *Hp*. Table 1 presents three parameter ranges related to the experiments described in this section. These ranges were defined analyzing the best rules obtained in our own experiments.

	NDS-R1	NDS-R2	NDS-R3
1^{st}Rule	**95.9%**	**96.2%**	**95.8%**
2^{nd}Rule	**95.7%**	**95.1%**	**95.2%**
3^{th} Rule	94.3%	94.5%	80.1%
4^{h} Rule	93.5%	91.8%	75.3%
5^{th} Rule	93.2%	87.5%	75.2%
#Rules>80%	15	6	3

Table 2. No diversity strategy environment: efficacy of the five best rules obtained in each experiment (measured in 10^4 initial lattices).

Each experiment was composed by 50 GA runs with the specification presented in last section. Table 2 presents the five best rules obtained in each experiment reported here. The name of the experiment is related to the range (1, 2 or 3) used and to the environment employed (in the case of Tab. 2, all experiments used no strategy to preserve the diversity). For example, the first column presents the five best rules found in the experiment performed using the environment with no diversity strategy and the range 1 to guide the genetic search (NDS-R1). The table also presents the number of good rules (here considered as efficacy above 80%) found in each experiment, out of the 50 GA runs.

All the experiments in Tab. 2 were able to obtain at least two rules with efficacy above 95%, which are better than the best one published in [14]. In fact, later we discover that the second best rule obtained in NDS-R2 was exactly the same rule as the one reported in [14] with an efficacy of 94.2%: EAC38AE8. A comparison between all the rules obtained in the experiments with efficacy above 95% is given in the end of this section. The best rule obtained in NDS-R2 experiment was the unique to go beyond the limit of 96%. Considering the three experiments reported in Tab. 2, it was possible to find 6 rules with an efficacy above 95%, being that the best result previously published for radius-2 rules was 94.2%.

The final experiments were performed using the environment with the strategy to preserve the diversity incorporated on it. The experiments were repeated in this new environment using the ranges in Tab. 1.

	DS-R1	DS-R2	DS-R3
1^{st}Rule	**95.8%**	**96.1%**	**96.2%**
2^{nd}Rule	**95.7%**	**95.3%**	**95.7%**
3^{th} Rule	**95.6%**	88,2%	**95.6%**
4^{h} Rule	**95.4%**	87,9%	**95.5%**
5^{th} Rule	**95.3%**	87,7%	94.9%
#Rules>80%	37	14	16

Table 3. Diversity strategy environment: efficacy of the five best rules obtained in each experiment (measured in 10^4 initial lattices).

Table 6.1 presents the five best rules obtained in each experiment reported here. For example, the first column presents the five best rules found in the experiment performed using the environment with the strategy to preserve diversity and the range 1 in Tab. 1 to guide the genetic search.

In general, the number of good rules (efficacy above 80%) obtained in each experiment have increased with the inclusion of the diversity strategy. The number of rules found with efficacy above 95% also increased and it was possible to find two rules with efficacy above 96%.

6.2 Comparing the best rules

Table 6.1 presents the hexadecimal code of the radius-2 CA rules found with efficacy above 95%. The table also presents the name of the experiment in which it was obtained. It is possible to see that some rules have been found in different experiments. For example, rule BD9EB000 was found in 3 experiments: NDS-FX1, DS-FX1 and DS-FX3**.** Rule EAC38AE8 was obtained in NDS-FX2 but as mentioned before it is exactly the same rule found in [14].

The actual evaluation of any CA rule performing an specific task considering a lattice of Ncells is given by the number of successes of this rule in all the 2^N possible configurations of this lattice. Historically, ST and DCT have been studied in lattices of size 149; it is impracticable to evaluate any rule in 2^{149} initial configurations. Therefore, it is common to use a bigger sample to have an estimate next to the actual evaluation, when comparing the performance of different rules for a task. Table 6.2 presents the evaluation of the best rules found in the experiments described in last section using a sample of 100.000 randomly generated lattices of 149 cells. This estimate is, therefore, more realistic than the estimate done to the end of each execution in which all of the rules have returned an efficacy above 95% in the respective experiments. Considering all the experiments described in this work performed using the heuristic based on sensitivity and neighborhood dominance heuristic, it was possible to find nine rules that overcome the best previously published rule (EAC38AE8), which presented in this more severe test an efficacy next to 95%. Five of them have an efficacy next to 95.5% and the best four have an efficacy next to 96%.

Rule	**Experiment**	**Efficacy in 10^5**
FD9E9042	**NDS-FX2**	**96.3%**
BDF68640	**DS-FX3**	**96.3%**
EA839AE8	**DS-FX2**	**96.2%**
AAE80AC0	**NDS-FX1**	**95.8%**
BD9EB000	**NDS-FX1 DS-FX1 DS-FX3**	**95.6%**
E8AFBCA8	**DS-FX3**	**95.6%**
BD9EB040	**DS-FX1**	**95.5%**
FFF28642	**DS-FX1**	**95.5%**
FD9EB002	**DS-FX3 DS-FX3**	**95.5%**
EAC38AE8	**Oliveira *et al.* [14], NDS-FX2**	**95.1%**

Table 4. Efficacy of the best rules found (measured in 10^5 initial lattices).

7 Final remarks

It was possible to find three rules with efficacy above 96%. The hexadecimal codes of the two best radius-2 rule evolved in our experiments are FD9E9042 and BDF68640. Both rules have an efficacy around 96.3%, measured in a sample of 10^5 lattices of 149 cells. As far as we know, they are the best radius-2 rules published to solve ST, using standard CA (synchronous update, periodic boundary conditions and deterministic rules). The insertion of the heuristic based on only two forecast parameters improved the performance of the evolutionary search. In [14], two more parameters had been used: absolute activity and activity propagation. One of the advantages on working with a lesser number of parameters is that it is easier to conduct experiments trying to refine the desirable bands, since there are a lesser number of variables to be adjusted. The incorporation of the strategy to preserve the population diversity also revealed efficient, increasing the number of good rules found in each experiment. Using this simple mechanism, it was possible to get more two rules with efficacy above 96% and several rules with efficacy above 95%. As future work, we intend to investigate the application of the four parameters in different combinations (individual, two by two and three by three). For this moment, we can advance that in experiments individually using sensitivity and neighborhood dominance, the improvement obtained with the incorporation of the heuristic is less significant than the observed with the both parameters.

8 Acknowledgements

Gina Maira B. de Oliveira thanks CNPq and FAPEMIG support.

References

[1] Andre, D.; Bennett III, F. and Koza, J.: Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. Proceedings of Genetic Programming 96. Stanford: Stanford University. (1996)

[2] Binder, P.: Parametric ordering of complex systems. Physics Rev. E. (1994)

[3] Culik II, K., Hurd, L. and Yu, S.: Computation theoretic aspects of cellular automata. Physica D, 45:357-378. (1990)

[4] Das, R., Crutchfield, J., Mitchell, M., Hanson, J.: Evolving globally synchronized cellular automata. Proc. of Inter. Conf. on Genetic Algorithms. (1995)

[5] Juillé, H. and Pollack, J.: Coevolving the ideal trainer: application to the discovery of cellular automata rules. Proc. of Genetic Programming Conference. (1998)

[6] Land, M. and Belew, R.: No perfect two-state cellular automata for density classification exists. Physical Review Letters, 74(25):5148. (1995)

[7] Langton, C.: Computation at the edge of chaos: phase transitions and emergent computation. Physica D, 42:12-37. (1990)

[8] Li, W., Packard, N.: The structure of elementary cellular automata rule space. Complex Systems, 4: 281-297. (1990)

[9] Mitchell, M., Hraber, P. , Crutchfield, J.: Revisiting the edge of chaos: evolving cellular automata to perform computations. Complex Systems, 7: 89-130. (1993)

[10] Mitchell, M.: Computation in cellular automata: a selected review. Nonstandard Computation. Weinheim: VCH Verlagsgesellschaft. (1996)

[11] Mitchell, M., Crutchfield, J., Das, R.: Evolving cellular automata with genetic algorithms: a review of recent work. Proc. of International Conference on Evolutionary Computation and Its Applications. (1996)

[12] Oliveira, G., de Oliveira, P. and Omar, N: Evolving solutions of the density classification task in 1D cellular automata, guided by parameters that estimate their dynamic behavior Proc. of Artificial Life VII: 428-436. (2000)

[13] Oliveira, G., de Oliveira, P. and Omar, N.: Definition and applications of a five-parameter characterization of one-dimensional cellular automata rule space. Artificial Life, 7(3):277-301. (2001)

[14] Oliveira, G., de Oliveira, P. and Omar, N.: Improving genetic search for 1D cellular automata, using heuristics related to their dynamic behavior forecast. IEEE Conference on Evolutionary Computation: 348-355. (2001)

[15] Oliveira, G., Bortot, J.. and de Oliveira, P.: Multiobjective evolutionary search for 1D cellular automata in the density classification task. Proc. of Artificial Life VIII: 202-206. (2002)

[16] Packard, N.: Adaptation toward the edge of chaos. dynamic patterns in complex systems. p.293-301. (1988)

[17] Sipper, M.: Computing with cellular automata: three cases for nonuniformity. Physical Review E, v. 57. (1998)

[18] Wolfram, S.: Universality and complexity in cellular automata. Physica D, 10:1-35. (1984)

[19] Wolfram, S.: Computation theory of cellular automata. Commun. in Mathmatical Physics, 96. (1984)

[20] Wuensche, A.: Classifying cellular automata automatically: finding gliders, filtering, and relating space-time patterns, attractor basins and the Z parameter. Complexity, 4 (3):47-66. (1999)

.

Development of CA model of highway traffic

Anna T. Lawniczak[1] and Bruno N. Di Stefano[2]

[1] Department of Mathematics and Statistics, University of Guelph,
Guelph, ON N1G 2W1, Canada,
alawnicz@uoguelph.ca
[2] Nuptek Systems Ltd,
Toronto, ON M5R 3M6, Canada,
nuptek@sympatico.ca, b.distefano@ieee.org

Abstract. We describe the results our development of a two dimensional CA highway traffic model capable of realistically simulating: a multi-lane highway with multiple entry and exit ramps situated at various locations, vehicle following, speed increment up to maximum speed, selectable on a per vehicle basis, lane change to pass or to go back to the right-most lane as it may be required by road rules in some jurisdictions, slowing down or stopping to avoid obstacles. In this paper we describe the model and hwCA.exe, the traffic simulator software packages in which the model has been implemented.

1 Introduction

Unidirectional single-lane traffic, with no intersections and no entry or exit ramps, is probably the simplest scenario of highway traffic that one can model. This is easily described by means of a one-dimensional CA with periodic boundary conditions. Rule 184 is the simplest one-dimensional CA model of this highway traffic scenario, Fig. 1: every particle, representing a vehicle, at every time step, moves one cell forward provided that next cell does not contain any other particle, see for instance [1] and [2]. If we assume that each cell may contain either a “1” or a “0” and that each “1” represent the presence of a vehicle and each “0” the absence of a vehicle, we can visualize vehicles moving forward at a constant speed in the absence of vehicles ahead of them, or otherwise stopping. According to Rule 184, at each time step, each vehicle tries to drive at a speed of 1 cell per time step, unless it is prevented from doing so by a vehicle immediately ahead of it, in which case the vehicle stops in order to avoid a collision.

The Truth Table of Rule 184 is shown in Tab. 1, where the first row shows all possible values at time t for three adjacent cells (c_{i-1} , c_i , c_{i+1}) and the second row shows the next state for the centre cell, at time t+1. Cells c_{i-1} and c_{i+1} are the neighborhood of cell c_i . Nothing can be said about cells c_{i-1} and c_{i+1} at time t+1 unless their entire neighborhood is known.

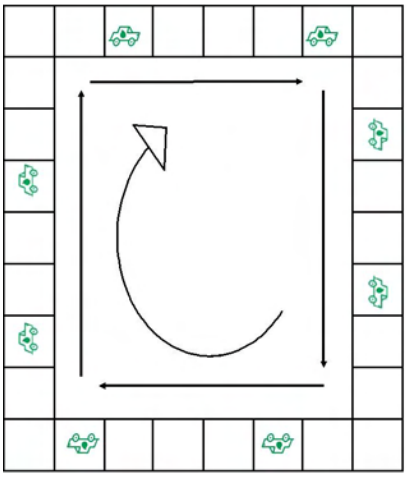


Fig. 1. Unidirectional single-lane traffic, with no intersections and no entry or exit ramps, i.e. Rule 184 with periodic boundary conditions.

Rule 184 is a rather simple traffic predictor, but it allows visualizing some of the well known patterns of real highway traffic, i.e.: platoons of moving cars separated by stretches of open road when traffic is light, and "traffic waves" moving upstream, that is in opposite direction to the movement of the vehicles, i.e., waves of stop-and-go traffic, when traffic is heavy. However, Rule 184 is of limited, if any, practical usefulness for traffic engineers and traffic managers when they try to realistically model traffic flow. Various authors have developed more complex CA models to more accurately simulate real traffic. See page 7 of [3] for a list of well established models and subsequent pages for a description of some of them. In 1992, Nagel and Schreckenberg proposed a stochastic traffic CA able to reproduce several characteristics of real-life traffic flows, [4]. Their model probably established CA as a valid method of modeling highway traffic flow. A large number of models followed, studying more realistic topologies, e.g. multilane expressways, intersections (e.g., [5] and [6]), urban areas (e.g., [5], [7], [8], [9], and [10]), etc.

We have developed a two-dimensional CA highway traffic model capable of realistically simulating: a multi-lane highway with multiple entry and exit ramps situated at various locations, vehicle following, speed increment up to maximum speed selectable on a per vehicle basis, lane change to pass or to go back to the

Table 1. Truth Table of Rule 184.

Time t	Current Pattern	111	110	101	100	011	010	001	000
Time t+1	Next state for centre cell	1	0	1	1	1	0	0	0

right-most lane as it may be required by road rules in some jurisdictions, slowing down or stopping to avoid obstacles. While our model is rather more complex than Rule 184, it can be obtained by relaxation and extension of Rule 184. In this paper we describe the model and hwCA.exe, the traffic simulator software packages in which the model has been implemented.

2 The model

We plan on using this model for practical traffic engineering applications, to estimate "travel time" between two access ramps, an entry ramp and an exit ramp, at different locations along the highway, once certain highway traffic parameters are known at certain points of the highway. Our concern is primarily with effects of flow and congestion on "travel time" through a long highway . We are interested in simulating simultaneously effects of phenomena studied individually by others (e.g., platoons of moving cars separated by stretches of open road when traffic is light, and "traffic waves" moving upstream, that is in opposite direction to the movement of the vehicles, when traffic is heavy, very small number of drivers exhibiting erratic behaviour and their effect on flow and congestion, etc). The difference between our work and published research previously conducted by others is that we model much longer highways, e.g. at least 500 km, and a much higher number of vehicles, e.g. realistic traffic conditions over several days.

We can have any number of lanes, but the most typical configurations are between 2 and 5 lanes. We are expected to be able to customize the model with the actual length of a real highway and with the actual number and location of entry ramps and exit ramps, modeling also their relative location. For instance, in Canadian highways an exit ramp is often before an overpass bridge, while the corresponding entry ramp is after such bridge. Our model is able to represent this and to show the distance between the two ramps. This is achieved by converting all distances into cell distances. Any cell or group of cells on the rightmost lane can be defined as an entry or an exit ramp.

Vehicles are generated at each entry ramp according to a probability predefined for that entry ramp. Each vehicle is instantiated with some properties, i.e., its vehicle type (i.e. car, bus, truck, truck with trailer, etc), its current speed (a variable updated at each time step to reflect driving conditions), its maximum allowable speed (constant through the trip, dependent on vehicle type but also on the type of driver, i.e. prudent, law abiding, careless driver, etc.), its entry ramp and its destination ramp (both constant through the trip), its time of entry (constant after the trip is started), its time of exit (measured at the end of the trip and used to measure various model performance indicators), its current cell and its current lane (dynamically updated at each time step). Moreover, there are some variables that are updated at each time step and help the vehicle to autonomously move through the highway, changing lane, passing other vehicles either on the left or on the right, as it may apply, accelerating, slowing down and braking if necessary. All these variables are measured in number of cells. They are: a velocity dependent safe distance, a look-ahead measure, a look back measure, and closest obstacle location. At every time step, the vehicle assesses the situation on the current lane, straight ahead, and immediately adjacent left lane, if existing, and the immediately adjacent right lane, if existing, both forward and backward. Once this information is obtained, the vehicle will act accordingly. A number of aggregate variables are calculated and stored for off line processing.

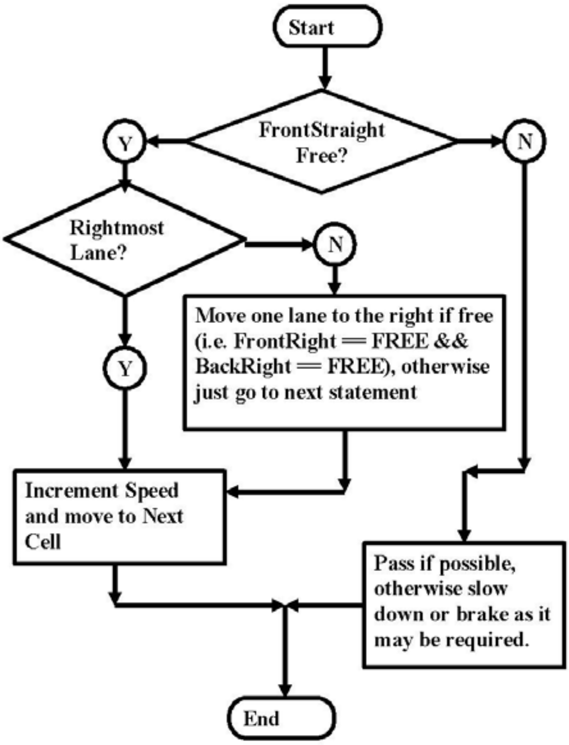

Start
FrontStraight Free?
Y
N
Rightmost Lane?
N
Y
Move one lane to the right if free (i.e. FrontRight == FREE && BackRight == FREE), otherwise just go to next statement
Increment Speed and move to Next Cell
Pass if possible, otherwise slow down or brake as it may be required.
End

Fig. 2. Simplified flow-chart of the algorithm used to implement the model.

A simplified flow-chart of the algorithm used to implement the model is shown in Fig. 2. The initialisation phase sets up the data structure used to store the highway. Vehicles are actually generated and instantiated during the simulation loop. Information about the location of each vehicle is stored twice in the model: in the highway data structure and in each vehicle object. The reason for this duplication is that the information stored in the highway data structure is global in scope, that is, accessible to all vehicles and is used by each vehicle to check the status of a possible destination cell, defined as a boolean variable either“EMPTY” or “NOT-EMPTY”, to avoid accidents (see Fig. 3 for an example of subset of the highway). The highway contains information about

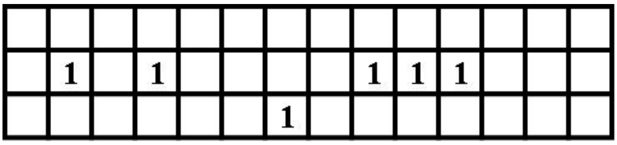


Fig. 3. Subset of the highway data structure showing a “1" where a vehicle is present and an empty space (a zero in the corresponding memory location) where no vehicle is present.

vehicles without knowing to which vehicle this information applies. Subsets of this “anonymized” aggregate information will be used in the future of our work to derive “travel time” information suitable for traffic engineers and traffic managers.

2.1 Neighbourhood

While traditionally we look at CA neighbourhoods of the von Neumann type (i.e., the four cells orthogonally surrounding a central cell on a two-dimensional square lattice) or of the Moore type (i.e., the eight cells surrounding a central cell on a two-dimensional square lattice), in our model we look at a vehicle position dependent neighbourhood like the one of Fig. 4.

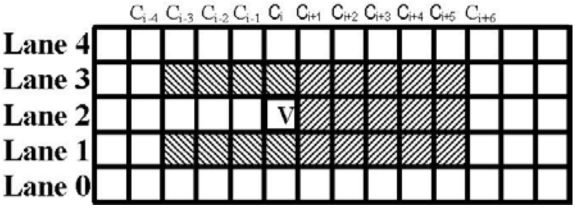


Fig. 4. Example of 5 lanes highway. The vehicle being considered is located at cell c_i on Lane 2. Its neighbourhood consists of two parts: the forward tilted lines show “Forward Neighbourhood”(consisting of cells c_{i+1}, c_{i+2}, c_{i+3}, c_{i+4}, and c_{i+5} on lanes 1, 2, and 3), while backward tilted lines show “Backward Neighbourhood” (consisting of cells c_i, c_{i-1}, c_{i-2}, and c_{i-3} on lanes 1 and 3).

If the vehicle under consideration is shown with a “V” on “Lane 2”, we distinguish a “Forward Neighbourhood” consisting of cells c_{i+1}, c_{i+2}, c_{i+3}, c_{i+4}, and c_{i+5} on lanes 1, 2, and 3 (forward tilted lines in Fig. 4), and a “Backward Neighbourhood”, consisting of cells c_i, c_{i-1}, c_{i-2}, and c_{i-3} on lanes 1 and 3 (backward tilted lines in Fig. 4). The exact number of cells making up a neighbourhood depends on the experiment being conducted. We define two parameters, LookAhead and LookBack, both measured in number of cell, to delimit the two subsets of the neighbourhood. The way the two parameters are set depends on the experiment being conducted. We have run experiments where all drivers are assumed to be prudent and used for both parameters the maximum allowable number of cells that can be covered in one time step, i.e. the maximum speed. We have run experiments where LookBack is equal to the maximum allowable number of cells that can be covered in one time step and LookAhead is a constant, speed independent, smaller number. We have also run experiments where LookBack is equal to the maximum allowable number of cells that can be covered in one time step and LookAhead is a smaller number dependent on the speed of the vehicle under consideration. The selection of the most realistic scenario is work currently in progress for which we are seeking feedback from traffic engineers. Examination of this neighbourhood, at each time step, results in each vehicle setting five boolean variables (i.e. FrontRight, FrontStraight, FrontLeft, BackRight, and BackLeft) to be used for decision making purposes for highway navigation at that time step.

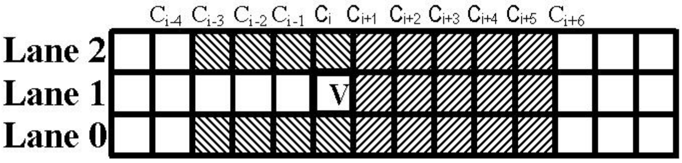

C_{i-4} C_{i-3} C_{i-2} C_{i-1} C_i C_{i+1} C_{i+2} C_{i+3} C_{i+4} C_{i+5} C_{i+6}
Lane 2
Lane 1
Lane 0
V

Fig. 5. Example of 3 lanes highway. The vehicle being considered is located at cell c_i on Lane 1. Its neighbourhood consists of two parts: the forward tilted lines show “Forward Neighbourhood”(consisting of cells c_{i+1}, c_{i+2}, c_{i+3}, c_{i+4}, and c_{i+5} on lane 0, 1, and 2), while backward tilted lines show “Backward Neighbourhood” (consisting of cells c_i, c_{i-1}, c_{i-2}, and c_{i-3} on lanes 0 and 2). From the neighbourhood point of view this scenario is equal to the one of a 5 lanes highway, Fig. 4.

The scenario of a 3 lanes highway is shown in Fig. 5. From the neighbourhood point of view this scenario is equal to the one of a 5 lanes highway, Fig. 4. However the situation is completely different in the case of a 1 lane highway, Fig. 6, where the “Backward Neighbourhood” is completely missing and the “Forward Neighbourhood” is limited to cells c_{i+1}, c_{i+2}, c_{i+3}, c_{i+4}, and c_{i+5} on lane 0. In this situation, only accelerating, decelerating, and braking are possible and the operation of the model is not too different from Rule 184, the only difference being that speeds other than 1 cell per time step are possible.The scenario of a 2 lanes highway is shown Fig. 7, where the vehicle is shown in the leftmost lane. In this case the neighbourhood consists of two parts: the forward tilted lines show “Forward Neighbourhood”(consisting of cells c_{i+1}, c_{i+2}, c_{i+3}, c_{i+4}, and c_{i+5} on lanes 0 and 1), while backward tilted lines show “Backward Neighbourhood” (consisting of cells c_i, c_{i-1}, c_{i-2}, and c_{i-3} on lane 0). Similarly in Fig. 8 we can see the case of a 2 lanes highway where the vehicle is shown in the rightmost

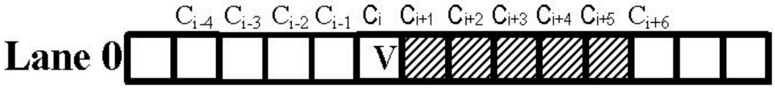


Fig. 6. Example of 1 lane highway. The vehicle being considered is located at cell c_i on Lane 0. Its neighbourhood consists the forward tilted lines, cells c_{i+1}, c_{i+2}, c_{i+3}, c_{i+4}, and c_{i+5} on lane 0.

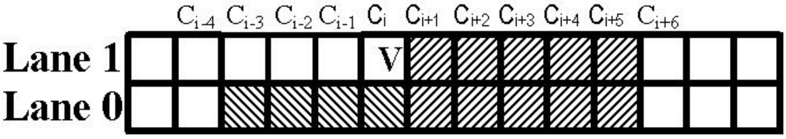


Fig. 7. Example of 2 lanes highway. The vehicle being considered is located at cell c_i on Lane 1. Its neighbourhood consists of two parts: the forward tilted lines show "Forward Neighbourhood"(consisting of cells c_{i+1}, c_{i+2}, c_{i+3}, c_{i+4}, and c_{i+5} on lanes 0 and 1), while backward tilted lines show "Backward Neighbourhood" (consisting of cells c_i, c_{i-1}, c_{i-2}, and c_{i-3} on lane 0).

lane. In this case the neighbourhood consists of two parts: the forward tilted lines show "Forward Neighbourhood"(consisting of cells c_{i+1}, c_{i+2}, c_{i+3}, c_{i+4}, and c_{i+5} on lanes 0 and 1), while backward tilted lines show "Backward Neighbourhood" (consisting of cells c_i, c_{i-1}, c_{i-2}, and c_{i-3} on lane 1).

2.2 Cell size and time step duration

To use this model for practical traffic engineering applications, we must assign a realistic length value, in metres, to each cell and a realistic time value, in seconds, to each time step of our simulation. Perusing the literature, it is possible to find that the most common length chosen for a cell is 7.5 m, see for instance [3], [4], [12]. This has been chosen because it corresponds to the space occupied by the typical car plus the distance to the preceding car in a situation of dense traffic jam. The traffic jam density is given by 1000/7.5m approximately equal to 133 vehicles/km, a figure deemed realistic by traffic engineers. We agree with the cell size of 7.5 m. This size allows us to model also busses, trucks, and trucks with trailers, just by assigning to them more than one cell. Thus, given the length of the highway in km, we take the integer part of (km x 1000.0/7.5) to determine the length of the highway in cells. After careful consideration, we have decided to assign the value of 3 seconds to each time step. Thus, the minimum speed of a vehicle advancing by one cell at each time step is equivalent to 9 km/h (that is, $7.5 \times 3600/3 = 7.5 \times 1200 = 9000$ m/h). This allows representing most realistic and legal speeds observed in Canadian highways, with a vehicle

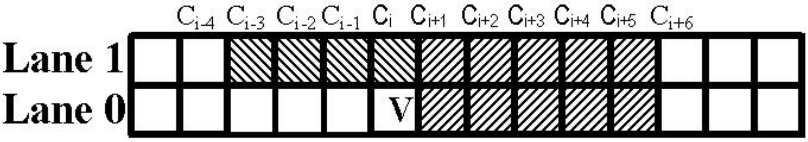


Fig. 8. Example of 2 lanes highway. The vehicle being considered is located at cell c_i on Lane 0. Its neighbourhood consists of two parts: the forward tilted lines show "Forward Neighbourhood"(consisting of cells c_{i+1}, c_{i+2}, c_{i+3}, c_{i+4}, and c_{i+5} on lanes 0 and 1), while backward tilted lines show "Backward Neighbourhood" (consisting of cells c_i, c_{i-1}, c_{i-2}, and c_{i-3} on lane 1).

advancing by a maximum of 11 cells per time step, that is, 99 km/h, as the speed limit is at 100 km/h. Nothing prevents modeling the presence of a few cars violating this speed limit. Lower number of seconds per time step would not allow sufficient granularity for our purposes. If higher resolution of the velocity is required, it would always be possible to assume that each digital time step is equivalent to a larger number of seconds. Other authors have preferred to choose a different cell size and different time steps duration for various reasons related to the experiments that they needed to conduct. For instance, Hafstein et al., [11], have chosen a cell size of m 1.5 and a time step of 1 second, with a speed resolution of 5.4 km/h. In their model a passenger car occupies between 2 and 5 contiguous cells, depending on the type of car and a truck occupies always 5 continuous cells.

3 The software

We called our software program hwCA.exe, for HighWayCA. The presence of "exe" in the name hwCA.exe is due to the fact that the early stages of design, coding, and testing have been carried on under Microsoft Windows XP. We designed it using object-oriented design methodology and coded it using the C++ programming language, using only features of the language that are compatible with the ISO/IEC JTC1/SC22/WG21 standard, to insure portability across various operating systems.

3.1 Private member variables of "Class Vehicle"

In what follows we show some of the private member variables of Class Vehicle. The static counters are global to the class (i.e., they belong to the class not to the object) and are incremented, as and if applicable, whenever an object constructor is executed and decremented, as and if applicable, whenever an object destructor is executed.

```
class Vehicle
{
private:
static long unsigned int VehicleCount;
static long unsigned int CarCount;
static long unsigned int BusCount;
static long unsigned int TruckCount;
static long unsigned int LongTruckCount;
VehicleType Type;
speed CurrentSpeed;
speed MaxSpeed;
cell EntryRamp;
cell DestinationRamp;
time TimeOfEntry;
time TimeOfExit;
cell CurrentCell;
cell SafeDistance;
cell Obstacle;
cell LookAhead;
cell LookBack;
lane CurrentLane;
bool FreeFlags[5];
};
```

3.2 Data types

We defined an enumerated custom data type called VehicleType, as follows:

```
enum VehicleType {CAR, BUS, TRUCK, LONGTRUCK};
```

It is used to define all possible types of vehicles known by the model. Different vehicle types result in different occupancy numbers, i.e., different number of cells occupied by each vehicle. The car is the only type of vehicle that occupies only one cell. For the purpose of evolution of the CA, the neighbourhood of each vehicle is defined always in relations to the back of the vehicle and the extra length is absorbed within the dimensions of the variable SafeDistance, which for all vehicles longer than one cell will always be represented by a bigger number than for vehicle occupying only one cell. At this stage, other vehicle types can be added only by modifying the definition of a number of switch statements, and recompiling the source code. We plan on allowing this modification from configuration file, at run time, for future versions of this program.

We have defined four custom data types as follows:

```
typedef long unsigned int cell;
typedef long unsigned int speed;
typedef long unsigned int time;
typedef short unsigned int lane;
//
// where
//
//#define ULONG_MAX     0xffffffffUL
//#define USHRT_MAX     0xffff
```

The reasons for defining these data types is to improve source code readability among project collaborators and to allow changing the actual "machine" data type in the future. Types "cell", "speed", and "time" allow specifying up to 4,294,967,295 items, well beyond our needs and expectations. For instance, given a cell size of 7.5 m, the maximum length highway that can be represented with this data type is long 32 million km and the maximum simulation time is 3.5 million hours. "Speed" is represented with this data type only because it is expressed in cells per time step and, thus, it must have the same data type as cells. Clearly, these dimensions are excessive. However, as we have considered using a much smaller cell size (e.g., 1.5 m) to capture the dynamics of such phenomena as abrupt braking, we felt that we should allow for the maximum possible dimensions. We plan on changing the definition of these custom data types to optimize the use of computing resources.

3.3 The navigation algorithm

The main simulation loop for each vehicle is equivalent to the following source code, that is, at each time step, the time is compared with the maximum duration of the simulation.

```
for(TimeStep = 0; TimeStep < CYCLES; TimeStep++)
{
DisplayHighway(TimeStep);
if(V.GetCurrentCell() < V.GetDestinationRamp())
V.Run(TimeStep);
}
```

(Pseudo-code showing the action taken at each time step for each object of type "Vehicle")

If the last cycle has been reached, the simulation is halted. If the program is running in "display mode", the utility to display the highway is updated. This is possible only for short highways or for segments of the highway. Under normal execution, the results of each time step are stored on disk, for the entire highway. Information about the current cell where each vehicle is located is retrieved. If this current cell is not yet close to the destination ramp, the navigation algorithm is run. If the destination ramp is within short distance, the vehicle is directed to the exit.

The actual Navigation Algorithm, for every vehicle, consists in (Fig. 9):

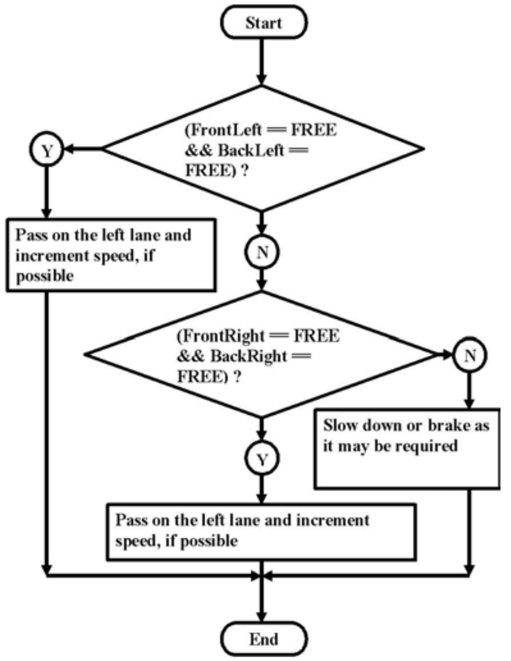

Start
(FrontLeft == FREE && BackLeft == FREE) ?
Y
Pass on the left lane and increment speed, if possible
N
(FrontRight == FREE && BackRight == FREE) ?
N
Slow down or brake as it may be required
Y
Pass on the left lane and increment speed, if possible
End

Fig. 9. Simplified FlowChart showing the Navigation Algorithm.

- acquiring information about the neighbourhood of the cell where the vehicle finds itself at the beginning of the time step
- calculating the five boolean flags (i.e. FrontRight, FrontStraight, FrontLeft, BackRight, and BackLeft) and the first obstacle straight ahead, if any
- actually navigating

The actual navigation, in the simplest case, consists in checking the flag FrontStraight. If it shows that the current lane is free, the vehicle checks if it is moving on the rightmost possible lane. If it is, simply it increments the current cell location as required by the current speed and increments its speed according to the preset acceleration, if it has not yet reached its maximum allowed speed. If the vehicle is not on the rightmost possible lane, checks if both FrontRight and BackRight are simultaneously "FREE". If they are, it moves to next lane to the right and increments the current cell location as required by the current speed and increments its speed according to the preset acceleration, if it has not yet reached its maximum allowed speed. If the vehicle determines that the FrontStraight boolean flag indicates that the current lane is "NOT-FREE", it first checks if FrontLeft and BackLeft are simultaneously "FREE". If they are, it moves to the next lane to the left and increments the current cell location as required by the current speed and increments its speed according to the preset acceleration, if it has not yet reached its maximum allowed speed. If FrontLeft and BackLeft are not simultaneously "FREE", that is, one or both of them is/are

"NOT-FREE", it checks if both FrontRight and BackRight are simultaneously "FREE". If they are, it moves to next lane to the right and increments the current cell location as required by the current speed and increments his speed according to the preset acceleration, if it has not yet reached its maximum allowed speed. If they are not, it slows down.

3.4 Slowing down or braking?

One of the difficulties of modeling and simulating highway traffic by means of CA is that, at every time step, only the neighbourhood is known. The only information about past history is what can be derived from the current state of the neighbourhood. Thus, from the point of view of the vehicle moving at full speed, a fixed obstacle ahead (e.g., a stalled vehicle) is not distinguishable from a slow moving vehicle ahead. We have designed the interaction among CurrentSpeed, CurrentCell, SafeDistance, Obstacle, LookAhead, and LookBack in such a way that a vehicle moving at full speed can come to a full stop within two or three time steps of detecting an obstacle, depending of its distance at the time of detection. If the obstacle is just a slower vehicle, the trailing vehicle can just slow down and, as soon as conditions allow for it, it can pass. If the obstacle is a permanent obstacle (e.g., a stalled vehicle), the moving vehicle will either pass on a different lane or stop and wait until it is safe to pass.

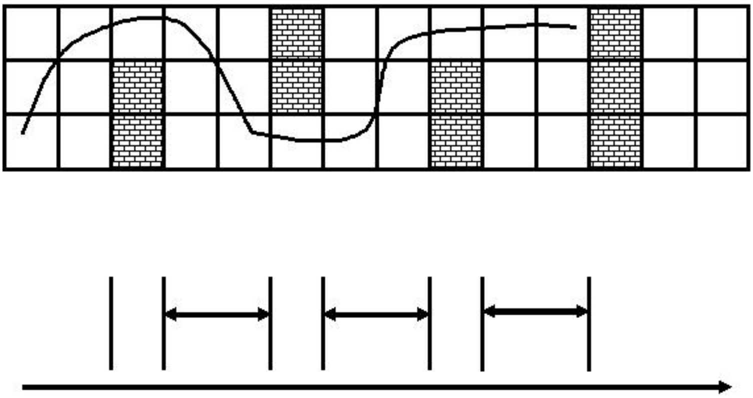


Fig. 10. Highly improbable, but not impossible, sequence of obstacles

In Fig. 10 we show a highly improbable, but not impossible, sequence of obstacles that has proven very challenging for the slowing down and braking algorithm. The leftmost "brick wall" occupying the rightmost and the centre lane is easy to overcome if no other vehicle is arriving on the leftmost lane. However, most algorithms have difficulties coping with the two brick walls on the leftmost and centre lane, i.e. the second set of obstacles from the left. The reason for this difficulty is that the five boolean flags (i.e. FrontRight, FrontStraight, FrontLeft, BackRight, and BackLeft) are all in the “NOT-FREE” value when the vehicle is in the “valley”, on the rightmost lane, between obstacles. Indeed, the neighbourhood, as shown in Fig. 4 or in Fig. 5 is too wide to accommodate this topology. Three solutions are possible:

- keeping track of the situation at previous time steps, thus violating the CA paradigm and moving toward agent based simulation
- introducing fuzzy logic control in the slowing down and braking algorithm
- making the dimension of the front neighbourhood a function of the vehicle velocity, thus showing FrontRight, FrontStraight, and FrontLeft to be “FREE” on a reduced neighbourhood, while they would appear to be “NOT-FREE” at higher speed

We favour the last alternative.

4 Conclusion

We presented an abstract of our research, the development of a two dimensional CA highway traffic model capable of realistically simulating: a multi-lane highway with multiple entry and exit ramps located at various locations, vehicle following, speed increment up to maximum speed selectable on a per vehicle basis, lane change to pass or to go back to the right-most lane as it may be required by road rules in some jurisdictions, slowing down or stopping to avoid obstacles. We plan on using this model for practical traffic engineering applications, to estimate travel time between two access ramps, an entry ramp and an exit ramp, once certain highway traffic parameters are known at certain points of the highway. Our concern is primarily with effects of flow and congestion through a long highway on travel time. The difference between our work and published research previously conducted by others is that we model much longer highways, e.g., at least 500 km, and a much higher number of vehicles, e.g. realistic traffic conditions over several days.

5 Future work

Work currently underway includes comparison with known published models for individual phenomena e.g. drivers exhibiting erratic braking behaviour, drivers unnecessarily changing lane, vehicle suddenly stalling while driving at full speed, and their effect on flow and congestion on the highway. After having verified each of these and other behaviours, we will simulated their impact on traffic by generating a number of vehicles exhibiting the behaviour under consideration based on probability distributions obtained from traffic engineers. Upon successful completion of this work we will estimate “travel time” between two access ramps, an entry ramp and an exit ramp, once certain highway traffic parameters are known at certain points of the highway. Given the size of the highways to be modelled, the number of vehicles, and the length of the simulations, we will use grid computing.

The software developed so far can be defined as an application framework, that still requires software skills when working on a new experimental scenario. We plan on making this software user friendly and to allow its use for people more comfortable with the application domain than with the software implementation domain.

6 Acknowledgements

A. T. Lawniczak acknowledges partial financial support from the Natural Science and Engineering Research Council (NSERC) of Canada. B. N. Di Stefano acknowledges full financial support from Nuptek Systems Ltd.

References

[1] Chopard B., Droz M., Cellular Automata Modelling of Physical Systems. Cambridge University Press (1998)

[2] Boccara N., Modeling Complex Systems. Springer-Verlag New York, Inc.(2004)

[3] Maerivoet S. and De Moor B.(2005). Cellular automata models of road traffic, in Physics Reports, vol. 419, nr. 1, pages 1–64, November 2005

[4] Nagel K., Schreckenberg M. (1992). A cellular automaton model for freeway traffic. J. Physique I 2, 2221–2229

[5] Chopard, B., Queloz, P.A., and Luthi, P.O. (1995). Traffic models of a 2D road network. 3rd European Connection Machine Users Meeting, Parma, Italy

[6] Chopard, B., Luthi, P.O., and Queloz, P.A. (1996). Cellular automata model of car traffic in two-dimensional street networks, J.Phys. A, Vol.29, pp.2325–2336

[7] Chopard B., Dupuis A., Luthi, P.O. (1997). A Cellular Automata Model for Urban Traffic and its applications to the city of Geneva. Proceedings of Traffic and Granular Flow. 1997

[8] Esser J., Schreckenberg, M. (1997). Microscopic simulation of urban traffic based on cellular automata. Int. J. Mod. Phys. C, 8: 1025–1036, 1997.

[9] Simon P. M., Nagel K. (1998). Simplified cellular automaton model for city traffic. Phys.Rev. E, 58: 1286–1295, 1998

[10] Dupuis, A. and Chopard, B. 2001. Parallel simulation of traffic in Geneva using cellular automata. In Virtual Shared Memory For Distributed Architectures, E. K¸hn, Ed. Nova Science Publishers, Commack, NY, 89–107.

[11] Sigurur F. Hafstein, Roland Chrobok, Andreas Pottmeier, Michael Schreckenberg, Florian C. Mazur (2004) A High-resolution cellular automata traffic simulation model with application in a freeway. Traffic Information System Computer-Aided Civil and Infrastructure Engineering 19 (5), 338 – 350.

[12] Larraga M.E., del Rio J.A., Schadschneider A. (2004). New kind of phase separation in a CA traffic model with anticipation. J. Phys. A: Math. Gen. 37 3769–3781.

Evolving robust cellular automata rules with genetic programming

Stefania Bandini, Sara Manzoni and Leonardo Vanneschi

Complex Systems and Artificial Intelligence (C.S.A.I.) Research Center
Department of Informatics, Systems and Communication (D.I.S.Co.)
University of Milano-Bicocca, Milan, Italy

Abstract. A framework for Genetic Programming (GP) based strategy to automatically evolve Cellular Automata (CA) rules is presented. Evolving Cellular Automata rules with Genetic Programming offers advantages, at least, in terms of readability and generalization ability. Both aspects are relevant advantages in case of real-life complex systems simulations: readability allows the final user to read and understand the solution proposed by the Genetic Programming based system, and simply adapt it to domain knowledge; generalization ability, the most important performance evaluation criteria for artificial learning systems, allows to apply generated rules to classes of configurations that share common patterns in the reference context. In particular, we describe the application of the proposed framework based on GP to evolve agent behavioral rules in a system of situated cellular agents (SCA) modeling pedestrian evacuation dynamics.

1 Introduction

The task of designing and producing Cellular Automata (CA) rules that exhibit a particular behavior is generally considered a very difficult one. Several solutions to automatically solve this problem by means of computer simulations based on Genetic Algorithm (GAs) [1, 2] were proposed (see for instance [3, 4, 5] and [6] for a review). The work of Sipper and coworkers represents a noteworthy contribution to the field (see for instance [7, 8, 9, 10]). In all those works the objective was to find CA rules that performed simple computational tasks or that could be used to simulate logic gates. In this work we propose a framework to evolve CA rules, based on Genetic Programming [11, 12] and apply it to learn behavior rules for a system of situated reactive agents (Situated Cellular Agents [13]) simulating pedestrian evacuation dynamics according to SCA, a generalization of CA approach [14].

The main motivation of adopting GP, instead of GAs, for evolving CA rules are related at least to *readability* and *generalization ability.* In GP, in fact, solutions that undergo the evolutionary process are, generally speaking, computer programs written, as much as possible (see Sect. 2 for a more detailed discussion), in a human comprehensible language. On the other hand, when generating

CA rules with GAs, potential solutions are normally CA transition rules represented as strings of characters as explained in [15], i.e. in a very compact and efficient, but at the same time rather cryptic way.

In this work, behavioral rules will be expressed like simple computer programs. The specific syntax in which these programs are expressed will be explained in Sect. 3. However the reader may simply refer to transition rules in a CA-based simulation model. According to SCA modelling approach, human crowds are described as system of autonomous, situated agents that act and interact in a spatially structured environment. Situated agents are defined as reactive agents that, as effect of the perception of environmental signals and local interaction with neighboring agents, can change either their internal state or their position on the structured environment. Agent behavior is specified in terms of L*MASS [16] formal language while execution environment for SCA agents is provided by SCA platform [17]. Interaction between agents can occur either locally, causing the synchronous change of state of a set of adjacent agents, and at–a–distance, when a signal emitted by an agent propagates throughout the spatial structure of the environment and is perceived by other situated agents (heterogeneous perception abilities can be specified for SCA agents).

Generalization is one of the most important performance evaluation criteria for artificial learning systems, in particular for supervised learning [18]. In recent work (discussed in Sect. 3.3), some interesting strategies have been proposed to improve GP generalization ability. In the pedestrian dynamics application context, it is particularly important that agent behavioral rules have a good *generalization* ability. This system performance can provide interesting advantages and improvements in terms of forecasting abilities and in the detection and prevention of critical situations (i.e. they should be able to work “reasonably” also for “new” situations, that have not been used to train the system and generate the model).

To the best of our knowledge, GP has never been used before to evolve CA rules, except for the noteworthy exception of [19], where GP is used to automatically generate an efficient rule for the *majority* (or *density*) task. For what concern the integrated adoption of MAS and GP, in [20] MAS self-organization cooperative mechanisms have been proposed to develop a programming language in which each instruction-agent tries to be in cooperative state with other instruction-agents and system environment. In [21], similarly to our proposal, a system of agents learn a communication protocols according to a GP approach.

2 Genetic programming

Genetic programming (GP) [11, 12] is an evolutionary approach which extends Genetic Algorithms (GAs) [1, 2]. As GAs, GP works by defining a goal in the form of a quality criterion and then using this criterion to *evolve* a set (also called population) of solution candidates (also called individuals) by mimic the basic principles of Darwin evolution theory [22]. Differently from GAs, the evolving GP candidate solutions are, generally speaking, computer programs. The quality of the individuals composing populations, or their likelihood of survival, is often called *fitness* and it is usually measured by an algebraic function called fitness function.

In synthesis, the GP paradigm breeds computer programs to solve problems by executing the following steps:

1. Generate an initial population of computer programs.
2. Iteratively perform the following steps until a given termination criterion has been satisfied:
   (a) Execute each program in the population and assign it a fitness value according to how well it solves the problem.
   (b) Create a new population by applying the following operations:
      i. Probabilistically select a set of computer programs to be reproduced, on the basis of their fitness (selection).
      ii. Create new computer programs by genetically recombining randomly chosen parts of two selected individuals (crossover), with probability p_c.
      iii. Copy some of the selected individuals, without modifying them, into the new population (reproduction), with probability $1 - p_c$.
      iv. Create new computer programs by substituting randomly chosen parts of some individuals with new randomly generated ones (mutation) with probability p_m.
3. The best computer program appeared in any generation is designated as the result of the GP process at that generation. This result may be a solution (or an approximate solution) to the problem.

Typical termination criteria are: a pre-determined number of iterations (also called generations) have been executed or a satisfactory solution has been found. Representation of GP individuals, initialization of GP populations, selection, crossover and mutation operators and a brief introduction to the theoretical foundation of GP are discussed below.

2.1 Representation of GP individuals

The most common version of GP, the one originally defined by Koza in [11], considers individuals as LISP-like tree structures. Thus, the set of all the possible structures that GP can generate is the set of all the possible trees that can be built recursively from a set of function symbols $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ (used to label internal tree nodes) and a set of terminal symbols $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ (used to label tree leaves). Each function in the function set $\mathcal{F}$ takes a fixed number of arguments, specifying its *arity*. Functions may include arithmetic operations ($+$, $-$, $*$, etc.), mathematical functions (such as `sin`, `cos`, `log`, `exp`), boolean operations (such as `AND`, `OR`, `NOT`), conditional operations (such as `If-Then-Else`), iterative operations (such as `While-Do`) and other domain-specific functions that may be defined. Each terminal is typically either a variable or a constant, defined on the problem domain. The function and terminal sets should be chosen so as to verify the requirements of *closure* and *sufficiency*: the closure property requires that each of the functions in the function set be able to accept, as its arguments, any value and data type that may possibly be returned by any function in the function set and any value and data type that may possibly be assumed by any terminal in the terminal set. The sufficiency property requires that the set of terminals and the set of functions be capable of expressing a solution to the problem.

2.2 Initialization of a GP population

Initialization of the population is the first step of the evolution process. It consists in the creation of the program structures that will later be evolved. If no *a priori* problem feature is known, populations are typically initialized with random individuals. The most common initialization methods in tree-based GP are the *grow* method, the *full* method and the *ramped half-and-half* method. Those methods are described in [11] and will not be discussed here.

2.3 The selection operator

At each generation, individuals have to be chosen for survival and mating. Many selection algorithms have been defined so far, the most popular ones being *fitness-proportional* or *roulette wheel*, *ranking* or *tournament* selection. Those methods, defined in [11], share two common properties: individuals with a better fitness have a higher probability of being selected compared to the ones with worse fitness and all individuals in the population must have a probability of being selected larger then zero. The selection algorithm used in the experiments presented in this paper is the tournament selection. It works as follows: a number of individuals, called *tournament size*, is chosen randomly and the one with better fitness is then selected. This procedure is repeated N times, where N is the population size. The tournament size allows to adjust selection pressure. A small tournament size causes a low selection pressure and a large tournament size causes a high selection pressure. This method is widely used in GP mainly because it does not require a centralized fitness comparison between all individuals in the population. This allows GP systems to save computational time and provides an easy way to parallelize the algorithm.

2.4 Crossover

The crossover (sexual recombination) operator creates variation in the population by producing new offspring that consist of parts taken from each parent. The two parents, that will be called T_1 and T_2, are chosen by means of one of the selection methods introduced above. Standard GP crossover [11] begins by independently selecting one random point in each parent (it will be called the crossover point for that parent). The crossover fragment for a particular parent is the subtree rooted at the node lying underneath the crossover point. The

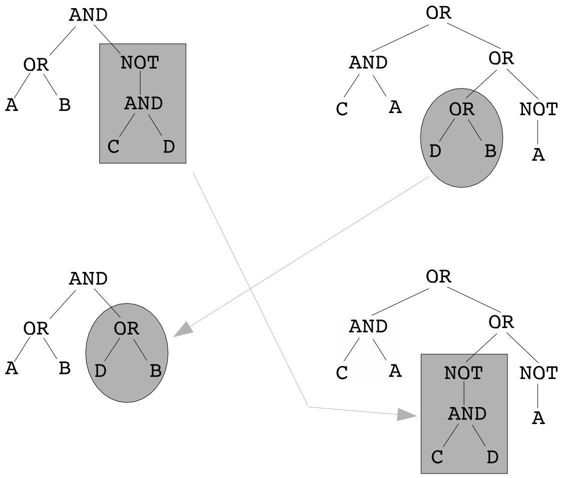


Fig. 1. An example of standard GP crossover. Crossover fragments are included into gray forms.

first offspring is produced by deleting the crossover fragment of T_1 from T_1 and inserting the crossover fragment of T_2 at the crossover point of T_1.

The second offspring is produced in a symmetric manner. Figure 1 shows an example of standard GP crossover.

Because entire subtrees are swapped and because of the closure property of the functions, crossover always produces syntactically legal programs, regardless of the selection of parents or crossover points.

2.5 Mutation

Mutation is asexual, i.e. it operates on only one parental program. Standard GP mutation, often called *subtree* mutation [11], begins by choosing a point at random within an individual. This point is called mutation point. Then, the subtree laying below the mutation point is removed and a new randomly generated subtree is inserted at that point. Figure 2 shows an example of standard GP mutation.

This operation, as it is the case for standard crossover, is controlled by a parameter that specifies the maximum depth allowed and limits the size of the newly created subtree that is to be inserted.

2.6 GP parameters

Once the GP user has decided the set of functions $\mathcal{F}$ and the set of terminals $\mathcal{T}$ used to represent potential solutions of a given problem, he still has to set some parameters that characterize evolution. A list comprising some of these parameters is the following one:

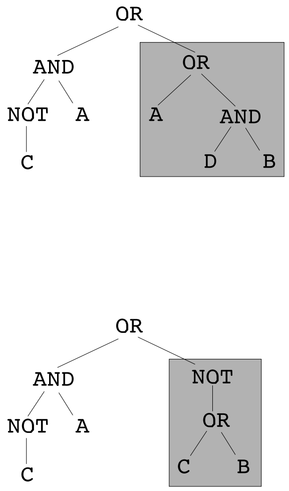


Fig. 2. An example of standard GP mutation.

- Population size.
- Stopping criterion.
- Technique used to create the initial population.
- Selection algorithm.
- Crossover type and rate.
- Mutation type and rate.
- Maximum tree depth.
- Presence or absence of elitism (i.e. survival of the best individual(s) into the newly generated population).

The setting of each one of these parameters represents in general a crucial choice for the performance of the GP system. Much of what GP researchers know about these parameters is empirical and based on experience.

2.7 GP theory

After reading the description of GP approach given so far, one question may come natural: why GP should work at all? This question can be made more precise by splitting it into the following ones: why the iterative GP process should allow to build solutions of better and better fitness quality? And why should it allow to find a solution that is satisfactory for a given problem? Or even better: what is the probability of improving the fitness quality of solutions along with GP generations? What is the probability of finding a satisfactory solution to a given problem? The attempt to answer these questions has been one of the main research activities in the GP field since its early years. Being able to answer the above questions surely implies a deep understanding of what happens inside a GP population along with generations. One may think of recording some numerical values concerning individuals of a population along with generations and of calculating statistics on them. Nevertheless, given the complexity of a GP system and its numerous degrees of freedom, any number of these statistical descriptors would be able to capture only a tiny fraction of the system's features. For these reasons, the only way to understand the behavior of a GP system appears to be the definition of precise mathematical models. Among others, references [23, 24, 25] contain a deep discussion of GP precise mathematical and probabilistic models and show a large number of interesting properties of GP, including its asymptotic convergence to a globally optimal solution.

3 Evolving with genetic programming pedestrian behavior rules in evacuation dynamics

In this section we will describe how the GP framework above introduced has been applied to improve a modeling and simulation framework for pedestrian dynamics (i.e. SCA4CROWDS [26]).

SCA4CROWDS is an ongoing research aiming at developing formal and computational tools to support the design, execution and analysis of models and simulations to study potentially complex dynamics that can emerge in Crowds and Pedestrian Dynamics as effect of physical and emotional interactions. Potential exploitations of this research are oriented to support the design and management of public spaces and events, and human sciences (i.e. social psychology) in theirs studies on crowds behavior [27]. SCA4CROWDS formal model that we developed as an extension of CA exploiting MAS advantages in modeling heterogeneous systems. The proposed approach has recently been presented within Pedestrian Dynamics modeling and simulation research area [26]. SCA4CROWDS provides a modeling framework based on an extension of CA (Situated Cellular Agents) where autonomous interacting agents share a spatial environment and behave according to individual behavioral rules and local available information. The main advantages in adopting an approach based on GP to generate agents' behavioral rules is towards experiments and robust analysis of performances of public spaces and structures (i.e. security and comfort).

3.1 SCA approach to pedestrian dynamics

According to SCA modelling approach, human crowds are described as system of autonomous, situated agents that act and interact in a spatially structured environment. Situated agents are defined as reactive agents that, as effect of the perception of environmental signals and local interaction with neighboring agents, can change either their internal state or their position on the structured environment. Agent autonomy is preserved by an action–selection mechanism that characterizes each agent, and heterogeneous MAS can be represented through the specification of agents with several behavioral types through L*MASS formal language. Interaction between agents can occur either locally, causing the synchronous change of state of a set of adjacent agents, and at–a–distance, when a signal emitted by an agent propagates throughout the spatial structure of the environment and is perceived by other situated agents (heterogeneous perception abilities can be specified for SCA agents).

SCA model is rooted on basic principles of CA: it intrinsically includes the notions of state and explicitly represents the spatial structure of agents' environment; it takes into account the heterogeneity of modelled entities and provides original extensions to CA (e.g. at–a–distance interaction).

According to SCA framework, the spatial abstraction in which the simulated entities are situated (i.e. *Space*) is an undirected graph of *sites* (i.e. $p \in P$), where graph nodes represent available space locations for pedestrians and graph edges define the adjacency relations among them (and agents' suitable movement directions). Each $p \in P$ is defined by $\langle a_p, F_p, P_p \rangle$, where $a_p \in A \cup \{\perp\}$ is the agent situated in p , $F_p \subset F$ is the set of fields active in p and $P_p \subset P$ is the set of sites adjacent to p. Pedestrians and relevant elements of their environment that may interact with them and influence their movement (i.e. *active elements of the environment*) are represented by different types of SCA agents. An agent *type* $\tau = \langle \Sigma_\tau, Perception_\tau, Action_\tau \rangle$ is defined by:

- Σ_τ: the set of states that agents of type τ can assume;
- $Perception_\tau : \Sigma_\tau \rightarrow W_F \times W_F$ **function** for agents of type τ: it associates each agent state to a pair (i.e. *receptiveness coefficient* and *sensitivity threshold*) for each field in F;
- $Action_\tau$: the behavioral specification for agents of type τ in terms of L*MASS language [16].

A SCA–agent is defined by a type τ, its current state ($s \in \Sigma_\tau$) and position in *Space* (*Space* is the undirected graph of sites $p \in P$, where P is the set of available positions for situated agents). Agent internal architecture is composed by three tasks that define the agent actual behavior (i.e. *Perception*, *Deliberation*, and *Action*) and two knowledge containers:

- ***Agent Knowledge Base (AKB)*** is the internal representation of agent state and of its local perceptions (e.g. set of fields active in its site, set of empty sites in its surrounding). The AKB updating can be the effect of agent actions or of a change in the agent environment perceived by the agent (e.g. an adjacent site becomes empty, a new field reaches the agent site or the agent moves to another site).
- ***Agent Action Set (AAS)*** collects the set of actions that are allowed to the agent in terms of L*MASS language. AAS is defined according to the agent type and cannot change during agent execution.

SCA approach does not specify a standard way to define agents' perception, deliberation and action. SCA platform (the execution environment for SCA-based models) has been designed in order to be incrementally extended to several execution strategies. In our experiments we adopted a synchronous–parallel execution method for the system (i.e. at each timestep each agent update their AKB perceiving their local environment and selects the action to be performed).

A conflict resolution strategy (indicated below as *site exchange*) has been introduced in order to solve deadlock situations when more than one agent has chosen the same destination site. The phase between perception and execution is *deliberation* that is, the component of an agent responsible of conflict resolution between actions, when multiple actions are possible. SCA approach has been already used for pedestrian dynamics, and some well–known phenomena (i.e. *Freezing by Heating* and *Lane Formation*) observed by social psychology empirical studies [28] have been successfully reproduced.

3.2 Evolving pedestrian behavior

In order to automatically generate rules for pedestrian dynamics during evacuation, we propose a GP setup inspired by the well-known artificial ant on the Santa Fe trail problem, first defined in [11] and later deeply studied in [25]. The simulation space has been defined as a regular grid populated by a set of agents. In a complex situation like evacuation, agents should coordinate and behave according to a common strategy, in order to obtain best system performances (e.g. total evacuation time or number of evacuated pedestrians). In this work, GP is proposed to evolve agent behavior towards a sort of navigation strategy for agents that maximizes a given evacuation objective. In the model, room exit is an given location in the environment (i.e. regular grid), typically located close to room borders, but unknown to agents (unless located in neighboring positions). Agents are scattered randomly in the grid, facing different directions and all of them have a limited perception ability on the surrounding environment (local view on its neighborhood only, as in CA). Even if the set of agents actions is quite simple (e.g. move ahead, lest of right), a number of different behaviors during each evacuation may be identified. The set of terminals used by GP for this problem are $\mathcal{T} = \{Right, Left, Move\}$ and corresponds to the actions an agent can perform according to this simplified SCA-based model of a pedestrian crowd: turn right by 90°, turn left by 90° and move forward in the currently facing direction. When an agent moves into the grid cell identifying the exit, we consider that it terminates its path. The set of functions may be $\mathcal{F} = \{IfObstacleAhead, Progn2, Progn3\}$. $IfObstacleAhead$ is a conditional branching operator that takes two arguments and executes the first one if and only if an obstacle is present in the case that is adjacent to the agent in the direction the agent is facing, and the second one otherwise. An obstacle may the the border of a path or the one of the grid, or even another agent. $Progn2$ and $Progn3$ are common LISP operators. $Progn2$ takes two arguments and causes the ant to unconditionally execute the first argument followed by the second one. $Progn3$ is analogous, but it takes three arguments, that are executed in an ordered sequence. An individual built with these sets $\mathcal{F}$ and $\mathcal{T}$ can be considered as a "program" that allows an agent to navigate the grid. An example of such program can be:

```
IfObstacleAhead(Prog3(Left, Left, Move),Move)
```

In the presented case study we considered as fitness function, the number of agents that have reached the exit before a given number of time steps. A time-out limit has to be fixed sufficiently small to prevent a random walk of the agents to cover the whole available area and thus finding the exit by hazard.

3.3 Genetic programming generalization

Next step in this research work will concern the application of known techniques to improve the generalization ability of GP to our framework for pedestrian evacuation dynamics. Generalization is particularly important for this kind of application. In fact, the same evacuation strategy may be suitable not only for one particular situation or space configuration, but a set of, so to say,"similar" or analogous situations. Many techniques have been developed in the last few years to increment GP generalization ability, as discussed in [29, 30]. A detailed survey of the main contributions on generalization in GP has been done by Kushchu in [29]. Another important contribution to the field of generalization in GP is due to the work of Banzhaf and coworkers; in particular, in [31] they introduce a new GP system called Compiling GP System and they compare its generalization ability with that of other Machine Learning paradigms. Furthermore, in [32] they show the positive effect of an extensive use of the mutation operator on generalization in GP using sparse data sets. In [33], Da Costa and Landry have recently proposed a new GP model called Relaxed GP, showing its generalization ability. In [34], Gagné and coworkers have recently investigated two methods to improve generalization in GP-based learning: 1) the selection of the best-of-run individuals using a three data sets methodology, and 2) the application of parsimony pressure to reduce the complexity of the solutions. A common agreement of many researchers is the so called minimum description length principle (see for instance [35]), which states that the best model is the one that minimizes the amount of information needed to encode it. In this perspective, preference for simpler solutions and overfitting avoidance seem to be closely related, given that it should be more likely that a complex solution incorporates specific information from the training set, thus overfitting it, compared to a simpler solution. But, as mentioned in [36], this argumentation should be taken with care as too much emphasis on minimizing complexity can prevent the discovery of more complex yet more accurate solutions. In [37, 38] Vanneschi and coworkers empirically show that multi-optimization on the training set can be in general thought of as a good strategy to improve GP generalization ability.

In synthesis, the main known techniques to improve GP generalization ability are:

- add noise to data (in this case, paths and space configurations);
- dynamically change the training set during learning;
- multi-optimization of the training set;
- automatically driving GP search towards regions of the search space composed by small and simple programs, in order to avoid overfitting.

As explained in [29] (where GP generalization on the artificial ant problem, that is similar to the application we propose here, is discussed), generalization can hopefully be obtained when the target function has “similar” characteristics on the training set as well as on test data. In case of pattern reconstruction applications, this mean that patters on which the system has been trained have to be as similar as possible to the ones the system has to handle in the testing phase. For obtaining this, the training phase has to be performed by submitting to our system many, and possibly different between them, “typical” space configurations.

4 Conclusions

Evolving Cellular Automata rules by means of Genetic Programming may prove particularly suitable, especially in simulations of real-life complex systems like pedestrian dynamics for evacuation. In fact, in those cases, a short set of simple instructions are usually given to crowds. Those instructions are not personalized and are often the same for different (although similar) situations. They can include advices about particular behaviour or directions to walk, or any other similar instruction and they often do not change in similar environments. Genetic Programming offers the following advantages in such situations: (1) readability, given that the proposed solutions are represented as human-like computer programs; (2) generalization ability, given that many techniques (some of which have been discussed in this paper) have been recently proposed to improve Genetic Programming generalization ability. These two points are particularly important, since point (1) allows the final user to read and understand the solution proposed by the Genetic Programming based system, and eventually modify it by hands according to some domain specific knowledge; point (2) allows to apply the same rules to classes of configurations, and not only to particular, well defined cases.

In the future, we plan to realize a Genetic Programming environment for robust learning of evacuation scenarios strategies, by automatically applying some of the most popular heuristics used to increment Genetic Programming generalization ability, such as adding noise to data, dynamically changing the training set during learning, multi-optimizing on the the training set and automatically driving GP search towards regions of the search space composed by small and simple programs, in order to avoid overfitting. This environment will be tested on real and complex evacuation scenarios, whose data will be collected by means of sensor networks or cameras.

References

[1] Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, Michigan (1975)

[2] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989)

[3] Smith III, A.: Real-time language recognition by one-dimensional cellular automata. Journal of Computer and System Sciences **6**(3) (1972) 233–253

[4] Ibarra, O., Palis, M., Kim, S.: Fast parallel language recognition by cellular automata. Theoretical Computer Science **41**(2-3) (1985) 231–246

[5] Ganguly, N., Maji, P., Dhar, S., Sikdar, B., Chaudhuri, P.: Evolving Cellular Automata as Pattern Classifier. Proceedings of Fifth International Conference on Cellular Automata for Research and Industry, ACRI 2002, Switzerland (2002) 56–68

[6] Mitchell, M., Crutchfield, J., Das, R.: Evolving cellular automata with genetic algorithms: A review of recent work. Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA,Äô96) (1996)

[7] Sipper, M.: Non-Uniform Cellular Automata: Evolution in Rule Space and Formation of Complex Structures. In Brooks, A, R., Maes, Pattie, eds.: Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV), MIT Press (1994) 394–399

[8] Sipper, M.: Co-evolving non-uniform cellular automata to perform computations. Physica D **92**(3-4) (1996) 193–208

[9] Sipper, M., Tomassini, M., Capcarrere, M.: Evolving asynchronous and scalable non-uniform cellular automata. Proceedings of International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97) (1997) 66–70

[10] Sipper, M.: Evolution of parallel cellular machines: The Cellular Programming Approach. Springer New York (1997)

[11] Koza, J.R.: Genetic Programming. The MIT Press, Cambridge, Massachusetts (1992)

[12] Vanneschi, L.: Theory and Practice for Efficient Genetic Programming. Ph.D. thesis, Faculty of Sciences, University of Lausanne, Switzerland (2004)

[13] Bandini, S., Manzoni, S., Vizzari, G.: SCA: a model to simulate crowding dynamics. Special Issues on Cellular Automata, IEICE Transactions on Information and Systems **E87-D**(3) (2004) 669–676

[14] Bandini, S., Manzoni, S., Simone, C.: Enhancing cellular spaces by multilayered multi agent situated systems. In Bandini, S., Chopard, B., Tomassini, M., eds.: Cellular Automata, Proceeding of 5th International Conference on Cellular Automata for Research and Industry (ACRI 2002), Geneva (Switzerland), October 9-11, 2002. Volume 2493 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2002) 156–167

[15] Wuensche, A.: Self-reproduction by glider collisions: the beehive rule. International Journal Pollack et. al (2004) 286–291

[16] Bandini, S., Manzoni, S., Pavesi, G., Simone, C.: L*MASS: A language for situated multi-agent systems. In Esposito, F., ed.: AI*IA 2001: Advances in Artificial Intelligence, Proceedings of the 7th Congress of the Italian Association for Artificial Intelligence, Bari, Italy, September 25-28, 2001. Volume 2175 of Lecture Notes in Artificial Intelligence., Berlin, Springer-Verlag (2001) 249–254

[17] Bandini, S., Manzoni, S., Vizzari, G.: Towards a platform for mmass–based simulations: focusing on field diffusion. Applied Artificial Intelligence **20**(4–5) (2006) 327–351

[18] Mitchell, T.: Machine Learning. McGraw Hill, New York (1996)

[19] Andre, D., Bennett III, F.H., Koza, J.R.: Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L., eds.: Genetic Programming 1996: Proceedings of the First Annual Conference, Cambridge, MA, The MIT Press (1996) 3–11

[20] George, J.P., Gleizes, M.P., Glize, P.: Basic approach to emergent programming - feasibility study for engineering adaptive systems using self-organizing instruction-agents. In: Third International Workshop on Engineering Self-Organising Applications (ESOA'05), Utrecht, Netherlands, 25-29 July 2005. (2005)

[21] Mackin, K., Tazaki, E.: Emergence of group behavior in multiagent systems using selftrained agent communication. In: Neural Networks, 1999. IJCNN '99. International Joint Conference on. Volume 4. (1999) 2241–2244

[22] Darwin, C.: On the Origin of Species by Means of Natural Selection. John Murray (1859)

[23] Poli, R., McPhee, N.F.: General schema theory for genetic programming with subtree swapping crossover: Part I. Evolutionary Computation **11**(1) (2003) 53–66

[24] Poli, R., McPhee, N.F.: General schema theory for genetic programming with subtree swapping crossover: Part II. Evolutionary Computation **11**(2) (2003) 169–206

[25] Langdon, W.B., Poli, R.: Foundations of Genetic Programming. Springer, Berlin, Heidelberg, New York, Berlin (2002)

[26] S.Bandini, S.Manzoni, M.L.Federici, S.Redaelli: A sca-based model for open crowd aggregation. In: Proceedings of the 4th International Conference on Pedestrian and Evacuation Dynamics, Wupperthal (D), Feb 2008. (2008)

[27] Still, G.K.: Crowd Dynamics. PhD thesis, University of Warwick, Warwick, (2000) http://www.crowddynamics.com/.

[28] Schreckenberg, M., Sharma, S.: Pedestrian and Evacuation Dynamics. Springer Verlag, Berlin (2002)

[29] Kushchu, I.: An evaluation of evolutionary generalization in genetic programming. Artificial Intelligence Review **18**(1) (2002) 3–14

[30] Eiben, A.E., Jelasity, M.: A critical note on experimental research methodology in EC. In: Congress on Evolutionary Computation (CEC'02), Honolulu, Hawaii, USA, IEEE Press, Piscataway, NJ (2002) 582–587

[31] Francone, F.D., Nordin, P., Banzhaf, W.: Benchmarking the generalization capabilities of a compiling genetic programming system using sparse data sets. In J. R. Koza *et al.*, ed.: Genetic Programming: Proceedings of the first annual conference, MIT Press, Cambridge (1996) 72–80

[32] Banzhaf, W., Francone, F.D., Nordin, P.: The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In W. Ebeling *et al*, ed.: 4th Int. Conf. on Parallel Problem Solving from Nature (PPSN96), Springer, Berlin (1996) 300–309

[33] Da Costa, L.E., Landry, J.A.: Relaxed genetic programming. In M. Keijzer *et al.*, ed.: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation. Volume 1., Seattle, Washington, USA, ACM Press (8-12 July 2006) 937–938

[34] Gagné, C., Schoenauer, M., Parizeau, M., Tomassini, M.: Genetic programming, validation sets, and parsimony pressure. In P. Collet *et al.*, ed.: Genetic Programming, 9th European Conference, EuroGP2006. Lecture Notes in Computer Science, LNCS 3905, Springer, Berlin, Heidelberg, New York (2006) 109–120

[35] Rissanen, J.: Modeling by shortest data description. Automatica **14** (1978) 465–471

[36] P.Domingos: The role of Occam's razor in knowledge discovery. Data Mining and Knowledge Discovery **3**(4) (1999) 409–425

[37] Vanneschi, L., Rochat, D., Tomassini, M.: Multi-optimization for generalization in symbolic regression using genetic programming. In Nicosia, G., ed.: Proceedings of the second annual Italian Workshop on Artificial Life and Evolutionary Computation (WIVACE), ISBN 978-1-59593-697-4 (2007)

[38] Vanneschi, L., Rochat, D., Tomassini, M.: Multi-optimization improves genetic programming generalization ability. In Thierens, D., ed.: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2007. Volume 2., ACM Press (2007) 1759

.

A simple cellular automata model for FX market forecasting

Michael Duane Archer

Molokai Technology Partners, USA
Email: Duane@FXpraxis.com
Home page: www.fxpraxis.com

Abstract. Since 1991 I have been to apply cellular automata (CA) to the problem of time series analysis generally and to forecasting Foreign Exchange (FOREX) prices, specifically [1]. Based on a number of backtests I have run, I now believe the answer to the question, "Is the market a Computer?' to be 'Yes!' This paper provides a very general discussion of my basic research using CA concepts for market price forecasting using what I call Trend Machine (TTM); a simplistic but perhaps effective conversion of 1D CAs to market forecasting. Specific algorithms and parameters are proprietary at this time. The paper is limited to making the conceptual connection between CA and markets and to an historical narrative of my efforts. A Simple Cellular Automata Model for FX Market Forecasting is written for a wide audience of CA theoreticians and market practitioners. I am neither a computer scientist nor a mathematician; I am a trader, since 1973, and researcher with passable programming skills in Visual Basic, Smalltalk (Dolphin), Prolog (Visual) and my current platform of choice, C#.net. I have my own programming team; some of whom have been with me since the mid-1980s. I developed the successful Jonathan's Wave expert system-neural network hybrid in the mid-1980s for commodity futures. The Trend Machine grew out of my conclusion AI methods — expert systems, neural networks, genetic algorithms — applied to the markets were fundamentally curve-fitting methods and functionally no different than conventional approaches such as stochastics, moving averages and oscillators. My interest in CA is purely practical — to find a consistent, risk-tolerable method of forecasting currency prices.

1 Introduction

My 'seed' idea for Trend Machines (TTM) came from reading, Three Scientists and Their Gods while doing consulting for the Winnipeg Grain Exchange in 1991 [2]. The book included an interview with and narrative about Ed Fredkin who brought CA out of the shadows in the 1970s. I had coincidently been working on a method for describing market prices in binary notation, 'Binary Charting' for a program Bar Chart Stat Pro which analyzes bar chart sequences ('1' = 'Up' and 'O' = Down as the basic syntax.) Like peanut butter and chocolate in the old Reese's commercial, the two ideas happily collided! The Eureka Moment came late one evening while my family was vacationing in Pensacola, Florida shortly after completion of the Winnipeg job. I woke my daughter, Brandy. We rushed to the local convenience store, purchased several packages of small wooden matches (much to the consternation of the store clerk) and experimented with a variety of a game described in Further Mathematical Diversions [3].

Some of the early TTM research, coding and back-testing was done with my Jonathan's Wave programming team of Richard Rowe (St. John's, Newfoundland, Canada), Antonio Riga (Toronto, Ontario, Canada) and James Bickford (Boulder, Colorado, USA). I lost contact with both Mr. Rowe and Mr. Riga in the mid-1990s; Mr. Bickford passed away in 2007. Curiously it was Mr. Rowe who very early on anticipated a problem I was not to acknowledge until much later — that my 'perfect world' TTM would run aground because of the Traveling Salesman P-NP Problem. My attempts to solve this issue cost me considerable time and effort but I did discover some interesting thoughts on P-NP which are beyond the scope of this paper.

I believe (after 17 years) I am very close to a significant breakthrough in algorithmic trading — but only the markets can, and will, tell the tale.

2 The Trend Machine — basic assumptions

Market forecasting methods have not been successful over the long term. Why? Techniques may be profitable for limited periods of time and/or specific environments, such as trending markets or trading markets. Success would be measured by profitability in diverse market conditions, for significant periods of time without dramatic 'draw downs' or deterioration of the capital base.

Two primary factors — directional movement and volatility — can reasonably describe the environment of any market. These primary market environments (MEs) plus two secondary environments — Thickness and Rhythm and a tertiary environment, Shape, describe every market [4].

Directional Movement (DM) is the net change in prices from a given price-time unit, x, to the next price-time unit, $x+1$. Volatility is the aggregate price change over that same period given a minimum fluctuation value. DM might be measured in a 90^0 arc from the horizontal chart axis representing zero to the vertical chart axis representing a real number of '4' or '10' or defining whatever precision is needed. V may be measured as a percentage of 100% and divided in kind as DM. '1-1' would represent a market with low directional movement and low volatility. '10-10' would represent a market with high directional movement and high volatility. ME effectively smoothes prices and has a wide variety of applications.

Most market forecasting techniques are linear. Proportionality is assumed between two or more factors. A change in factor 'X' causes a proportionate and constant change in factor 'Y'. This is an assumption about markets implied in all pattern recognition techniques.

But pattern recognition suffers some major theoretical flaws. Patterns (or input), no matter how sophisticated, identified as similar may lead to widely divergent output. If ‘X’ then ‘Y’ is not true of market data. Traders may recognize this from the ’sensitivity to initial conditions’ of chaos theory. Nor is the converse true (If ‘Y’ then ‘X’). Even if ‘X’ then ‘Y’ is true, the converse is a logical fallacy — modus ponens.

Internalizing the pattern search does not help. Expert systems, neural networks and even genetic algorithms suffer the same pattern recognition flaw as do moving averages, relative strength and stochastic tools. At best they are non-linear tools seeking linear information.

The author built the first expert system for market forecasting in the 1980s, Jonathan’s Wave. In AI Expert magazine [5] Harvey Newquist described it as a hybrid expert system neural network. In futures, it generated a 48% annual return with a zero expectation of a 50% drawdown and negatively correlated with other trading systems at the time [6]. I wrote about Jonathan’s Wave twice for Futures magazine [7].

At an AI conference in 1991 I suggested these difficulties to the attendees and elicited a very negative response [8].

Complexity theory studies processes resistant to linear analysis. Economies, ecologies, physical and life sciences are all areas with processes falling under the complex category. Information theory, game theory, encryption and coding and even corporate decision-making may be added. These processes seem to possess an internal organization not describable, much less predictable with linear methods of logic, mathematics or statistics. The markets appear also to be complex processes.

Some non-linear processes, such as the markets, generate time-series data as a by-product.

- Complex processes share several features; different types of complexity are defined by dominant features. In addition to resistance to linear methods and tools the most important are:
- Complex processes cannot be broken down and analyzed piece-by- piece. Nor can they be ’reverse engineered.’ Study of the effect does not lead to knowledge of the cause. This is the finis for pattern recognition.
- Complex systems are sensitive to initial conditions. A very small change in initial conditions may lead to enormous changes latter in the process. This is the defining characteristic of closed-end processes associated with chaos theory.
- Complex processes often manifest abrupt changes from one kind of behavior (’continuity’) to another kind of behavior (’discontinuity’). Catastrophe theory studies such open-ended systems. Why does a sideways market ’suddenly’ break out into a trending market?
- Recursive/Fractal: The behavior of complex systems is often played out at many different levels, all with great structural similarities. That is, each level is a macrocosm of the level below it and a microcosm of the level above. Another way to state this is that the processes are dimensionally recursive.

- Very simple system elements and rules create surprisingly diverse and complex behavior.
- Complex processes self-organize information. They appear to adapt by generating new, internalized rules. This self-organization can be likened to a computer program calling a sub-routine or a function with a new value. Complex systems behave as computational entities!

Consider the analogous characteristics of price markets: stocks, commodities, currencies, including the performance of investment fiduciaries:

- Linear methods have failed. The markets cannot be analyzed piece by piece. Nor does the study of the behavior (prices or information patterns) yield information about causes.
- The markets, as open-ended processes, are sensitive to minute input changes — the buy order that turns the market.
- Trading ranges suddenly erupt into trends, and visa versa.
- Two simple elements or rules create the complex tapestry of the markets: 'buy' and 'sell.'
- Hourly, daily, weekly and monthly bar charts of a stock or commodity exhibit many structural similarities. Without labeling it would be impossible to decide the time frame of a chart.
- I believe self-organization is the Major Market Force.

Discounting, the smoothing of spreads and the gradually lessening of the effectiveness of new market indicators are examples of the market utilizing self-organization as an adaptive 'immune system.' Discounting is a minor example of the market self-organizing input (buy and sell orders) into prices (output).

Is it possible to forecast the behavior of complex processes, especially those that generate time-series data as a by-product, especially stocks, currencies (FX) and commodities?

Since 1991 I have explored the investment markets, particularly currencies and commodities, as a complex process. Despite some initial excitement, chaos theory didn't catch. Although markets do appear to be recursive, that function does not seem sufficient to generate forecasts of any real accuracy. I will discuss inter-dimensional recursiveness later in this paper. Markets are open-ended and chaos applies primarily to closed-end processes. Catastrophe is a much better try, but catastrophe does a better job describing than predicting.

Markets self-organize buy and sell orders into prices. It does this by the mechanism of an internal algorithm determining how much UP a buy order creates and how much DOWN a sell order creates. The ongoing flow of behavior (prices) is non-linear. This behavior is impossible to predict without some knowledge of the underlying algorithm for the market in question.

The markets may not be predictable using computers, per se, but may be very predictable as computers — as computational entities. Pattern recognition tools and linear methods study the market-computer's screen dump. What needs to be studied is the program or algorithm generating the prices I see on the market-computer screen. I have been looking at the markets 'inside-out' (or 'outside-in') just as pre-Copernican astronomers took the Earth as the center of the solar system. The Market in the Machine!

The market is the pattern, and it is continually unfolding across the frontier of 'now.'

The data by-product of any market encrypts useful information about the underlying process.

Many processes generate data as a by-product. The market is a complex process and prices are the primary by-product. Assumption: the by-product carries information about the underlying process. Please keep this in mind: prices are the by-product of an underlying process.

Algorithmic Forecasting (the term as first used by this author in 1992) attempts to duplicate the algorithm that creates or *describes* at least a part of the self-organizing behavior of a data producing complex process. Once this algorithm is found, the process can be modeled (on a regular computer!) and ran through the barrier of 'now' into the future.

Today, generally, the terms 'algorithmic forecasting' and 'algorithmic trading' reference any automated trading system or scheme in all market classes — securities, commodities and FOREX.

Viewing the market as a computer or computational entity solves two great mysteries of technical analysis:

1. How is information about prices transmitted from the past to the future?
2. Do past prices influence future behavior?

The markets behave as an algorithm with a feedback loop. Prices at T[ime]1 (input) are at least partially organized by a market's specific algorithm. This yields new prices at T2 (output) that in turn becomes input for the next iteration of the algorithm, leading to prices (output) at T3. Buy and sell orders become the algorithmic parameters.

In sixteen years of research I've been able to draw two conclusions: a) the underlying algorithm is a major factor in prices, but not the only factor, b) the degree to which it is a factor ebbs and flows resulting in our conclusion that 'The markets may be 'busted' from time to time for relatively short periods of time.'

Are yet-to-be-entered orders Acts of God, or can they be predicted?

Clearly I cannot predict new orders perfectly, in advance. But the market-as-computer tells us the algorithm of a market creates a sieve, or template, through which all orders must pass. In programming, a function may return a different value each time it is called, but the value is delimited by the parameters of the function.

It may be possible to find ‘basins of attraction’ or ‘areas of criticality’ — areas in price and time to which new orders are pulled or attracted. Limits in commodities are an artificially created basin of attraction. This is an assumption of TTM v2.x, please see the sections below.

Can the markets, or any other data-producing complex process be modeled for algorithmic forecasting to ‘tease out’ self-organizing, algorithmic behavior?

The model I developed uses cellular automata (automaton, singular) as the basis of a market-as-computer model. A typical cellular automaton may be visualized as a grid of squares, or cells. Imagine an infinite chessboard, or chart grid.

The first cellular automata gedanken experiments of the famous John Von Neumann. He developed many details of CA after a suggestion of Stanislaw Ulam. My first exposure to CA was in 1971 when I attended a lecture by Professor Ulam at the University of Colorado in Boulder. (Chess Tradition in Man’s Culture, hosted by Professor Eugene Salome; lecture by Professor Ulam — Chess and Mathematics.)

A two dimensional cellular automata (2D-CA), the ’infinite chessboard’, is composed of five elements:

1. The cell universe
2. The individual cell(s)
3. The cell state
4. The cell neighborhood
5. The cell rules or algorithm.

The Cell Universe is a continuous arrangement of cells; the infinite chessboard.

A Cell is an individual unit, usually square or rectangular (but occasionally polygon tiled) within the cell universe; ‘King 5’ or ‘Queen Bishop 15,000’ on the infinite chessboard.

The Cell State refers to the condition of a cell. Usually a cell has only two states — ON or OFF (colored, uncolored). It is possible for cells to have multiple or even ‘fuzzy’ states.

The Cell Universe State may refer to either the current state of all the cells in a CA, or a collection of all prior universe states (generations) and the current state (generation) in some models.

The Cell Neighborhood is a selected group of cells surrounding a given cell. Cell neighborhoods may be either local or remote. Cells in a local neighborhood are physically adjacent to the cell. Remote cells are not adjacent. Two typical local neighborhoods are the Von Neumann neighborhood (side adjacent cells) and the Langton neighborhood (side adjacent and diagonal adjacent cells). Neighborhoods may also be state local or state remote.

Cell Rules or Algorithms are sets of instructions telling a given cell what state to assume in the next generation in response to the state condition of its neighbors in the current generation. Cell rule sets are usually simple, but may be large in number.

Neighborhood 'state conditions' may be very simple or enormously complex. Meta-level CA's may be used to define state conditions. I have spent more time exploring state conditions than any other factor in my lengthy and exhaustive research. In a typical CA only the state conditions of the neighbors in the current generation are used. But it is possible for cell rules (and neighborhoods) to rely on several previous generations (i.e., state remote). In one experiment I used a CA algorithm to determine which neighborhoods to use a Trend Machine.

A CA is the sequential iteration of the cell rules resulting in changes to each successive generation of the cell universe.

As they progress from generation to generation CAs evolve into one of four types of behavior:

1. Death — all cells die and go to OFF
2. Stasis — a finite loop of cell universe states repeats. It may take several thousand iterations of the cell rules for a CA to display Stasis.
3. Random — cell behavior changes and fluctuates without rhyme or reason. Random CAs almost always evolve into Death or Stasis.
4. Life — the CA generates new and interesting behaviors. Cells are born and die; groups prosper and falter. Complexity increases and the CA exhibit self-organization.

It seems impossible to predict the type of CA from the cell rules without actually running the CA. Two extremely similar rule sets may lead to life and death. This, in fact, is the quietus for TTM v1.x, as below.

CAs are examples of computational entities. Cells states act as both input (data) and (program). Cellular automata are being used today to study many complex processes; especially those for which self-organization is a primary characteristic. The emerging science of A-life grew out of early CA experiments. CAs of Type 4 mimic the complete topology of life: birth, death, reproduction, adaptation and mobility.

Who is the market participant who has not had occasion to exclaim that the market seems often to have '*a life of its own*?'

3 Work and efforts to 1999 — TTM v1.x

A Trend Machine (TTM) uses a mapping calculus to convert a CA into a market model for the purpose of algorithmic forecasting. By modeling market price data as a CA it is easy to search for the algorithms that self-organize buy and sell orders into market prices.

The TTM calculus converts each element of a CA into a Trend Machine component:

CA	TREND MACHINE
cell state	price up / price down
cell neighborhood	price history
cell universe	the market
cell rules	algorithm engine/ca-base
previous state	previous price
current state	current price
next state	future price

In a simple 2D Trend Machine (TTM) each iteration of the CA moves forward in time (from left to right) and the cell universe is the collection of cell universe states. A basic 2D calculus would integrate rules forcing iterations into a single vertical column causing the CA to mimic a bar chart showing the range of high to low prices over a specified period of time. Another involves mapping formations such as the gliders found in Life to price chart formations.

TTM and CAs may be constructed in one, two, three or n-dimensions.

The Algorithm Engine (AE) is roughly equivalent to the inference engine in an expert system. The algorithm engine consists of three components:

1. The basic structural design for building algorithms
2. The variables that can be altered within the AE
3. The seed or initial cell universe state

Re-writing refers to the method used to convert market prices (or other time-series data) to TTM states and back again into market prices. Re-writing may be, and often is, inter-dimensional. Two dimensional CAs may be ‘rewritten’ to generate binary, on-dimensional CAs. CAs may also be cross-dimensional, as data output from ’sound’ may be re-written to a ‘visual’ dimension. In this sense, algorithms as dimensionally recursive!

In a one dimensional TTM (1D-TTM) there is no grid. Cell states are simply ‘1’ or ‘0’. In the basic model a ‘1’ state would mean prices UP over the previous state and ’0’ would mean prices DOWN from the previous state. There is only one cell in each iteration of the cell universe. A sequence of cell states becomes a binary string. This effectively converts the market data ‘by-product’ into CA ready-to-use material!

I have worked primarily with 1D-CAs, but 2D, 3D and n-dimensional models (especially cross-dimensionality) have some fascinating possibilities.

It is theoretically possible to describe a higher dimensional TTM or CA in a one-dimensional scheme. Encoding the information in the binary string does this. (It has been previously demonstrated that a one-dimensional CA is a Turing Machine and all CAs of dimensionality $1 + n$ can be described in 1 dimension, see The Emperor’s New Mind by Roger Penrose.)

Indeed, since the limitations of generating uniqueness from a binary string are definitional-limited by the length of the seed, the most useful algorithms for The Trend Machine come from 2D-CAs which have been re-written to 1D, although this researcher has found a method to generate unique 1D CAs. It is, unfortunately, probably impossible to predict the output of these algorithms without actually running them. Unique binary strings are also more plentiful at 2D and 3D levels.

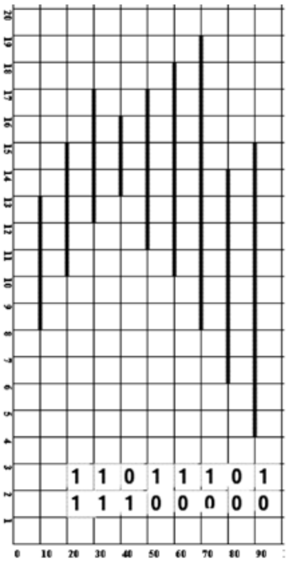


Fig. 1. Basic ordinal diagram. Each ‘1’ and ‘0’ may represent either an ordinal value (Up/Down) or a cardinal value (Up/Down with fixed, specific numeric value to each unit), in some re-writing schemes.

The binary string output of a 1D-TTM may be re-written (converted to/interpreted as market data) in many different ways. For example, see Fig. 1:

- Last (Close Only)
- Point and Figure
- Bar-Range / Horizontal Cluster
- Bar Range / Vertical Cluster
- Candlestick
- Pretzel
- Mercury[1]

Current versions of TTM use an ordinal ’stacked semaphore’ configuration of High, Low and Last prices:

HIGH	10011101010110110100110101000
CLOSE	10010000111101110110101010100
LOW	01101111010101011110100010101

[1] Charting techniques developed by FXpraxis.

Specific cell-states, neighborhood rules, functions (parameters) and transition rules are currently proprietary.

There are many different possible rewrite schemes — even for binary. While 'High, Low, Close' here means 'from the previous unit' CAs may and do take rules from other than the nearest (or, dominant) neighbor. Such distant neighbor seeding can significantly increase the number of possible unique algorithms. The string representations of '1' and '0' may also be either ordinal or cardinal values, simply 'UP' or 'UP 1.5 Units.' Converting to cardinal would probably increase effectiveness but cannot be done until the Caching Problem (see below) is solved.

Using my Market Environments methodology [8] has allowed my work to remain in the ordinal domain.

I have already mentioned higher dimensions can be encoded in binary strings. It is also possible to re-write directly between dimensional models.

A RAM or Representational Automata Machine is a CA modeled in n-dimensional state-space. A RAM might model all the factors of a 3D-CA (price, volume, open interest) and add, for example, volatility and an inter-market relationship. Like 3D-CAs RAMs allow for such specific factor mapping.

Four steps are involved in making a forecast using a Trend Machine:

1. Convert the market data (prices) to a binary format. Ordinal data conversion smoothes the data and leaves only directional movement and limited volatility information.
2. (a) Use the Algorithm Engine to find or build the algorithm most closely fitting the data.
   (b) Search the CA-base for the closest match to the data.
3. Convert the binary CA string to prices using the appropriate format.
4. Concatenate the string to generate a forecast.

Forecast 2a is a perfect world. I have not been able to build a robust algorithm engine that would generate and algorithm from a data string output. Given the data can you systematically construct the algorithm (backward construction)? Probably not. Given the algorithm can you predict the output without actually running the algorithm itself (forward construction). Perhaps. All my TTM v1.x models have instead used a CA-base, 2B.

The current TTM CA-base consists of strings from 28 unique 'methods' with multiple permutations, parameter sets and seeds within each model. I am constantly on the hunt for new methods in 1, 2, 3 or n dimensions or in cross dimensional space.

The hunt for algorithms leading to unique strings is fascinating — and a bit addictive. I have at least managed to classify algorithm types and have a general sense of what will definitely not work — lead to uniqueness. I seek a method for determining what definitely will work.

An example of a extremely simple algorithm: Calculate π to x precision. Convert the odd numbers to '0' and the even numbers to '1.' For example: 3.14159265358979323846 → 3.01000110001000010111

4 Significant issues

Back-testing is used to find many secondary parameters, such as how long a CA-Base string is needed to generate a forecast — and how long is the forecast meaningful.

The most significant issue has been the failure to find a general purpose Algorithm Engine (AE) that would take a binary string and find appropriate algorithms to generate that string and forecast-concatenations thereof. It appears neither a forward-chaining AE (predicting a string from an algorithm without running it) and a backward-chaining AE (mechanically finding an algorithm to match a string) are possible. But, on the side, I continue to toy with the idea simply because of its enormous attractiveness.

In 1999 a friend and fellow FX trader, Sylvestor Torini, suggested a 'CA-Base.' The idea, quite simply — develop a database of binary CA strings. At specific time increments (5-minute, 1-hour, etc) the program would convert market data to the appropriate binary template and search the CA-Base for a perfect match. For example, a string of n_1–n_{500} where in n_{501}–n_X would be reconverted to price format and used as the forecast.

It quickly became apparent a CA-Base of any value would be HUGE. Because of the 'sensitivity to initial condition premise' it is only a perfect match that will do.

I saw immediately computer processing speed would not solve the problem. Given Moore's Law I calculated it would be many years before the Base could be analyzed with even 5-minute price data! I began to investigate the Traveling Salesman, P-NP Problem. Though not identical, the two problems have remarkably similarities.

I spend two years looking for caching algorithms of many varieties; a method for using the free time between increments to pre-sort the CA-Base. Having worked with a group of Go programmers in the UK I spent some effort on Monte Carlo sampling but that floundered quickly for me. By early 2006, I was somewhat stymied and TTM v.1.x seemed to have run its course.

I spent more time on the P-NP Problem and considered the possible solutions types: A complete solution, an approximation solution and a special-case solution. Run out on the complete solution thread, knowing that an approximation solution was a poor fit for a CA model, I was left with the special case solution to work with and ponder.

5 Current efforts — TTM v2.x

It was at lunch with my lead programmer, the late Jim Bickford, in June of 2006 that I devised the strategy currently in place — and which shows substantial promise.

Jim labeled the idea 'One-Dimensional Catastrophe Topologies' (ODCT) — accurate, but something of an ontological and semantic stretch. The method is not dissimilar to the Bar Chart Stat Pro program (FXpraxis, 2002) which analyzes sequences of price bars of four types — Inside, Outside, Bear and Bull.

Instead of using the entire CA-Base I am extracting only strings which represent or lead to

1. A high percentage of '0s' [downtrend],
2. A very high percentage of '1s' [uptrends] ,
3. Nearly equal '0s' and '1s' [trading markets], and
4. some 'special case' string 'formations' that offer real-time trading opportunities.

Even this represents a rather largish Base, but I am constantly pruning and filtering to make it manageable and applicable to smaller and smaller time frames. In the quest to drop time-frame size, my most current work (February 2008) involves using and searching as a function of Market Environments (ME) of directional movement (DP) and Volatility (V) instead of prices. This may, in turn, lead to a simplified CA-Base.

It is important to recognize profiting in a market does not require the forecasting of exact prices. Forecasting Directional Movement and Volatility is adequate to the task, even with a precision of 20%-25% against an arc of 90^0 on a price chart.

The most significant advantage of the current approach — the ODCTs may be searched offline and not real-time while the markets are running. A still numerically significant but much smaller portion of the CA-Base needs to be searched real-time, see example in Fig. 2.

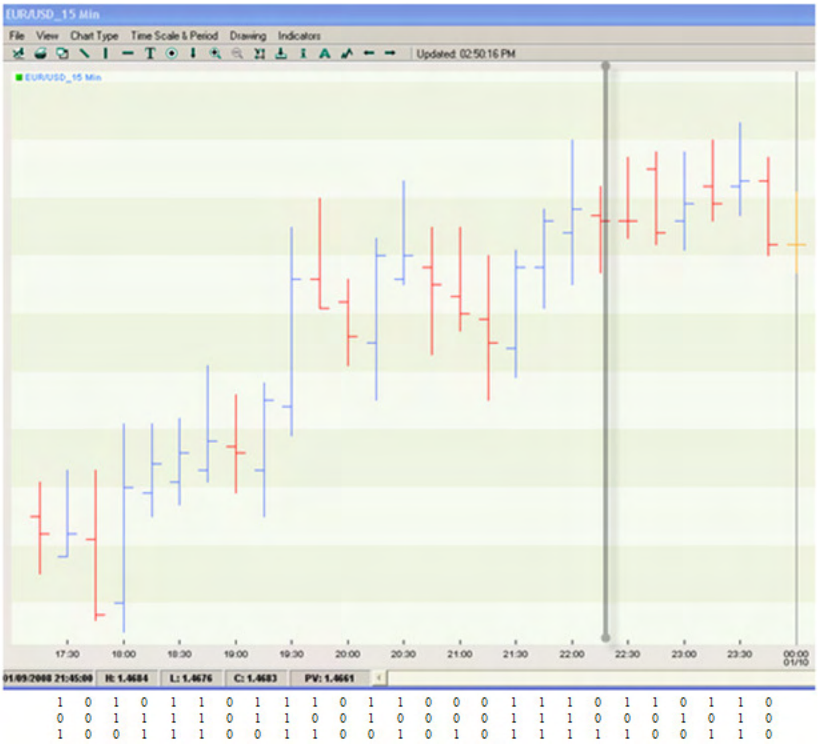
6 Price-time frames or increments

It became apparent several years ago there was a high correlation between length of time frame and (a) number of forecasts generated and (b) accuracy of forecasts. The shorter the time frame used, the more forecasts are made and the higher they rank statistically. I don't have enough data on time frames of under 15-minutes but it seems intuitive this is an exponential function, or nearly so. One point — it is clear the longer the time frame the more 'noise' there is in the data. The theory of high frequency trading may, indeed, be correct [10]!

Today I am able to use 15-minute data and am confident further work will, using allow increments below 1-minute by using (1) an ODCT CA-Base to analyze offline, (2) ME to smooth the incoming data and (3) a redesigned CA-Base to take full advantage of ODCT and ME.

7 TTM v2.x — The trade-off

In theory, an Algorithm Engine or complete CA-Base search will hit all available trading opportunities. But even a very limited scan of a CA-Base using a variety of caching methods (which will, of course, already filter out many trades) using



01/09/2008 21:45:00 H: 1.4684 L: 1.4676 C: 1.4683 PV: 1.4661

```
1 0 1 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 0 1 1 0 1 1 0
0 0 1 1 1 1 0 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0
1 0 0 1 1 1 0 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 0 1 1 0
```

Fig. 2. Comparison of prices for the EUR/USD currency pair (Euro to US Dollar) against a match from the ODCT offline CA-Base. The bars to the left of the vertical line are the ODCT match. The bars to the right of the vertical line are the forecast by concatenating the ODCT match. The program was set to seek only perfect matches of a minimum of 20 pre-forecast units.

1-hour data cannot match the number of trades generated by ODCT with 15-minute data with a full scan.

I continue to search the (still expanding) CA-Base for ODCTs and am in the process of categorizing and further analyzing the latter. My next (and final?) step in the process is to integrate the components into a real-time algorithmic trading program. My programming team is investigating NinjaTrader, `www.ninjatrader.com`, for this application. Ninja is particularly suitable because its scripting language, NinjaScript is a C# subset.

8 Explorations

When one ponders a problem for 17 years many interesting side roads are discovered; if only one could walk them all!

I continue to dream of a general purpose algorithm engine.

I am exploring a CA-Base populated with binary genetic algorithm strings and using GA as a filter/caching methodology.

I can see more conventional forecasting methods such as expert systems utilizing the TTM methodology.

Other time-series models and problems may benefit from the TTM methodology. I have briefly investigated manufacturing process control as one possibility.

I have spend almost all of my time on a 1D TTM model. The possibilities for 2D, 3D and nD models is staggering. With limited time and money I decided early just 'not to go there.' Of particular interest to me is a 3D-TTM using Price, Volume and Open Interest in commodity futures. But many of my algorithms were found by using 2D CAs and collapsing them into 1D. Although the methods are testable it is doubtful they would pass full academic scrutiny.

I have not discussed my research or methods in generating unique binary strings and consider it to be proprietary to my work.

9 Discussion

I continue to believe a method of forecasting market prices with extreme accuracy and very low risk parameters is possible. If it is possible it is only so in a very limited set of market conditions and over very short periods of time. Thus, the quest to operate using very fine price-time increments of 1-minute or less.

If such a method is to be found it will be in the use of non-linear tools and methods. Of those I have concluded cellular automata most closely matches the ontology of markets. 35 years in the markets tell me linear methods are 'closed' and have little or no predictive value.

I am currently back-testing TTM v2.41 and anticipate trading at least a prototype model in 2009. I have attempted to attract interest for final research and trading funds from a number of major players in the FX space to no avail. There is a curious dÈj‡ vu for me to the reception Jonathan's Wave received at the same stage in development, circa 1985. A company with a vision finally 'took a chance' and did very well because of the success of that program. Curiously, today, the large hedge funds are known as the world's ultimate risk takers — but in fact, are very conservative, in-the-box thinkers. I consider their quantitative analysis tools, for example, to be deficient and have developing my own methods, based on my Market Environments methodology.

I hope to make TTM v2.x updates available on my website[2] and am happy to discuss my research (within reason) with others.

References

[1] Archer, M. ECML '98 Workshop Notes. Application of Machine Learning and Data Mining in Finance. Is the Market a Computer? Technische Universitat Chemnitz, Chemnitz, Germany, April 1998.

[2] www.fxpraxis.com

[2] Wright, R. Three Scientists and Their Gods. New York, TimesBooks, 1988.

[3] Gardner, M. Further Mathematical Diversions. New York, Penguin Books, 1977.

[4] Archer, M. Back-testing and Market Environments. Currency Codex, `www.fxpraxis.com`, 2006.

[5] Newquist, H. Somewhere Over the Rainbow: Using AI to Get to Wall Street. AI Expert, July 1988.

[6] Baratz, M.. Jonathan's Wave (Review). Boston, Managed Account Reports, 1989.

[7] Archer, M. An Expert System for Trading Commodities. Conference on Artificial Intelligence. Chicago, University of Chicago, 1989.

[8] Archer, M. The Artificial Intelligence Approach to Expert Trading Systems. Futures, July 1986. Archer, Michael. How Expert Systems Compare with Toolbox Programs. Futures, May 1987.

[9] Archer, M. Market Environment Applications. Currency Codex, `www.fxpraxis.com`, 2004.

[10] Jorda, O. et al. Modeling High-Frequency FX Data Dynamics. February, 2002.

.

DDoS attack detection using entropy of packet traffic in CA like data communication network model

Anna T. Lawniczak[1], Hao Wu[1] and Bruno N. Di Stefano[2]

[1] Department of Mathematics and Statistics, University of Guelph,
Guelph, ON N1G 2W1, Canada,
(alawnicz,wuh)@uoguelph.ca
[2] Nuptek Systems Ltd,
Toronto, ON M5R 3M6, Canada,
nuptek@sympatico.ca, b.distefano@ieee.org

Abstract. We study applicability of information entropy for the detection of distributed denial-of-service (DDoS) attacks in a packet switching network (PSN) model by monitoring entropy of packet traffic at selected routers in a network. Given a certain PSN model setup (i.e., topology, routing algorithm, and source load) a "natural" entropy profile, a sort of "fingerprint" of the given PSN model setup, characterizes normal PSN model operation, i.e. normal packet traffic. Whenever, the entropy deviates from its "natural" profile in significant way, it means that some packet traffic anomaly is emerging. Detecting shifts in entropy in turn detects anomalous traffic and, in our virtual experiment, a ping DDoS attack.

1 Introduction

Distributed denial-of-service (DDoS) attacks are network-wide attacks that cannot be detected or stopped easily. They change "natural" spatio-temporal packet traffic patterns, i.e. "natural distributions" of packets among routers. Thus, they change "natural" entropy or "fingerprint" of normal packet traffic. Detecting shifts in entropy of packet traffic monitored at selected routers may provide means for detecting anomalous packet traffic. We explore this possibility and study entropy based detection of DDoS attacks in a packet switching network (PSN) model. Our model is a modification of a CA like PSN model of the Network Layer of the 7-Layer OSI Reference Model and its C++ simulator, Netzwerk, developed by us in [1], [2], [3], and [4]. Using this model we study entropy based detection of ping type DDoS attacks.

Our paper is organized as follows. First, we provide background information about attacks affecting PSNs in general and DDoS in particular. We briefly summarize DDoS attacks detection methods not based on information entropy and discuss how the DDoS attacks change the character of the flow of packets in the PSNs. We introduce the concept of entropy and explain why and how it can be used to detect anomalous traffic, thus, potentially to detect DDoS attacks. We briefly describe our existing abstraction of PSN and its simulator, Netzwerk and explain how they have been customized to model a ping type DDoS attacks. Next, we introduce the definition of entropy functions used by us in detection of DDoS attacks in our virtual experiments. We present selected simulation results and our conclusions.

2 Attacks on packet switching networks — short review

The packet switching technology was conceived by Paul Baran as a way to communicate in the aftermath of a nuclear attack, as a sort of resilient command and control network [5]. Though no implementation of packet switching network has ever had to undergo the test of its ability to withstand a nuclear attack the Internet, one of the best known applications of packet switching technology, is constantly under attacks of different types, e.g.: intrusion (unauthorised access), capturing (packet tapping), phishing (an "attempt to criminally and fraudulently acquire sensitive information", [6]), computer worms (self-replicating computer programs covertly sending copies of themselves to other nodes over the network [7]), computer viruses ("computer programs able to copy themselves, eventually mutating, and infecting other computer without permission or knowledge of the user", [8]), denial of service attack ("an attempt to make a computer resource unavailable to its intended users.", [9]) , and many others. Purpose and scope of these attacks are different, ranging from the commercial to the political domain, and often serve only the self actualization of the perpetrators.

We investigate how information entropy can be applied for detection of distributed denial-of-service (DDoS) attacks in a CA like packet switching network (PSN) model by monitoring entropy of packet traffic at selected routers in a network. The most common implementation of denial-of-service (DoS) is the distributed DoS (DDoS) attack. The attack is "distributed" because the attacker carries on his/her actions by means of multiple computers, located at various network nodes, called "zombies", and almost always controlled in a covert and surreptitious way without any knowledge of their legitimate owners. Thus, DDoS attack is a network attack that explores asymmetry between network-wide resources and local capacity of the target (victim) machine to process incoming packet traffic. In DDoS attack the victim machine and its neighbouring nodes become quickly saturated with buildup of intended congestion, such that they cannot respond to legitimate traffic any longer [9].

Our study focuses on the type of DDoS directing a huge number of "ping" requests to the target victim of the attack. This type of attack exploits the "Internet Control Message Protocol" (ICMP). "Ping is a computer network tool used to test whether a particular host is reachable across an IP network", [10]. "It works by sending ICMP 'echo request' packets to the target host and listening for ICMP 'echo response' replies. Ping estimates the round-trip time, generally in milliseconds, and records any packet loss, and prints a statistical summary when finished.", [10]. By issuing a huge number of ping 'echo requests' from a very large number of "zombies" spread all over the network, it is possible to cripple the target victim and make it unable to conduct any network activity other than answering the ping 'echo requests' and, eventually, rendering it so overloaded that it will come to a standstill. Various types of ping based DDoS attacks exist, including, but not limited to the popular "ping flood", [11], and "ping of death", [12]. Computer experts often quoted as an example of these type of attacks the Mafiaboy attacks against Amazon, eBay and other big sites on February 2000 [13], [14]. The Mafiaboy attacks caused millions of dollars damage.

3 Review of DDoS attacks detection methods

Since DDoS attacks are network-wide attacks they cannot be detected or stopped easily. The detection of DDoS attacks is usually studied from three different perspectives: 1) near the victim; 2) near attack sources; and 3) within transit networks, see [15] and reference therein. In these approaches packet data (e.g., headers, aggregate flows, and correlations) are analysed with the aim of distinguishing normal traffic from attack packets. However, it is a very difficult task because in many DDoS attacks packets are "normal-looking" and the existing methods are not accurate enough to distinguish between normal and attack packets. For early detection of DDoS flooding attacks Yuan and Mills in [15] proposed monitoring macroscopic network-wide effects of shifts in spatio-temporal patterns of packet traffic. Their proposed method is based on studying spatio-temporal correlations of packet traffic monitored at a number of observation points in a network. They have shown in [15] that given a sufficient number of observation points one can infer a shift in packet traffic patterns for larger areas outside the observation routers. Thus, their proposed method of macroscopic network-wide monitoring can provide cues when more detailed analysis of packet traffic should be commenced against potential DDoS attack.

4 Entropy based detection of DDoS attacks

In our study inspired by [16], we use information entropy, i.e. a measure of uncertainty, to detect DDoS attack by monitoring entropy of packet traffic at selected routers in a network. The thermodynamic concept of entropy provides a measure of the disorder of a system, i.e. a measure of the degree to which the probability of the system is spread out over different possible states. The thermodynamic entropy theory has been design to describe the configuration of a system based on a series of outcome probabilities in such a way that high entropy relates to high probability of outcome and low entropy relates to low probability of outcome. Thus, there is the equivalence between thermodynamic entropy and amount of uncertainty one has about an outcome. Namely, frequently occurring events provide less information then infrequently occurring events, and this links thermodynamic and information entropy. If the information entropy of the outcome/system is low we are less ignorant about the uncertainty about the outcome/system and this may be used in the detection of DDoS attacks.

Since DDoS attacks are purposely created by humans they must affect natural "randomness" and "natural structure and order" of packet traffic under normal conditions. Thus, DDoS attacks must affect the entropy of "normal packet traffic" and by detecting the shifts in packet traffic entropy one may in turn detect anomalous traffic. By building first a "fingerprint", i.e. a profile of entropy of packet traffic under normal conditions one may establish a baseline against which the shifts in entropy can be measured for DDoS attack detection purposes.

In our study of entropy based detection of DDoS attacks we use modification of our PSN model of the Network Layer of the 7-Layer OSI Reference Model and its C++ simulator, Netzwerk [1], [2], [3], [4]. In what follows we describe briefly our PSN model and its modification.

5 CA like PSN model and its modification

Our PSN model [1, 3], like in real networks is concerned primarily with packets and their routings; it is scalable, distributed in space, and time discrete. It avoids the overhead of protocol details present in many PSN simulators designed with different aims in mind than studying macroscopic network-wide dynamics of packet traffic flow and congestion. We view a PSN connection topology as a weighted directed multigraph L where each node corresponds to a vertex and each communication link is represented by a pair of parallel edges oriented in opposite directions. In each PSN model setup all edge costs are computed using the same type of *edge cost function* (*ecf*) that is either the *ecf* called ONE (ONE), or *QueueSize* (QS), or *QueueSizePlusOne* ($QSPO$). The *ecf* ONE assigns a value of "one" to all edges in the lattice L. Since this value does not change during the course of a simulation this results in a static routing. The *ecf* QS assigns to each edge in the lattice L a value equal to the length of the outgoing queue at the node from which the edge originates. The *ecf* $QSPO$ assigns a value that is the sum of a constant "one" plus the length of the outgoing queue at the node from which the edge originates. The routing decisions made using *ecf* QS or $QSPO$ rely on the current state of the network simulation. They imply adaptive or dynamic routing where packets have the ability to avoid congested nodes during the PSN model simulation. In our PSN model, each packet is transmitted via routers from its source to its destination according to the routing decisions made independently at each router and based on a *minimum least-cost criterion.* The PSN model uses *full-table routing*, that is, each node maintains a routing table of least path cost estimates from itself to every other node in the network. The routing tables are updated at each time step when the *ecf* QS or $QSPO$ is used. They do not need to be updated for the static *ecf* ONE, see [1], [3]. We update the routing tables using *distributed routing table update* algorithm.

In our simulations to study DDoS attacks we use a version of PSN model in which each node performs the functions of host and router and maintains one incoming and one outgoing queue which is of unlimited length and operates according to a *first-in, first-out policy*, see [3] for other options. At each node, independently of the other nodes, packets are created randomly with probability λ called *source load*. In the PSN model all messages are restricted to one packet carrying only the following information: time of creation, destination address, and number of hops taken.

In the PSN model time is discrete and we observe its state at the discrete times $k = 0, 1, 2, \ldots, T$, where T is the final simulation time. At time $k = 0$, the set-up of the PSN model is initialized with empty queues and the routing tables are computed. The time-discrete, synchronous and spatially distributed PSN model algorithm consists of the sequence of five operations advancing the simulation time from k to $k+1$. These operations are: (1) *Update routing tables*, (2) *Create and route packets*, (3) *Process incoming queue*, (4) *Evaluate network state*, (5) *Update simulation time.* The detailed description of this algorithm is provided in [1], [3].

For our study of DDoS attacks we modified the above described PSN model to allow modeling a PSN containing one victim computer and a user defined number of "zombies" either located at specified nodes or located at random. Start and end of attack time can be specified separately for each zombie. As in most real life cases, "zombies" continue to carry on their normal jobs during the attack, i.e. they act also as sources, destinations, and routers of legitimate data transfers. However, each "zombie" also sends a packet to the victim at each time step of the simulation.

6 Discussion of simulation results

Netzwerk simulator provides information about a number of network performance indicators, i.e. *number of packets in transit*, *average number of packets in transit*, *average delay time of packets delivered*, *average path length*, *average speed of delivery*, *throughput*, and *critical source load*, [3]. The impact of DDoS attacks on network performance indicators and spatio-temporal packet traffic dynamics will be discussed elsewhere. Here we focus on entropy detection of DDoS attacks.

We calculate entropy of packet traffic passing through monitored routers of PSN model as follows. Let M be a set of N monitored routers or nodes. The set M may include all network nodes except “zombies” and the victim. We index all routers in the set M by the parameter i (i.e., $i = 1, \ldots, N$). We denote by $q(i, k)$ a number of packets at the outgoing queue of a router i at time k. At each time k we calculate probability density function $p(i, k)$ of packets queuing at a router i of the set M as follows

$$p(i, k) = \frac{q(i, k)}{\sum_{i=1}^{N} q(i, k)}.$$

We calculate *entropy function of packet traffic* monitored at routers of the set M as

$$H(M,k) = -\sum_{i=1}^{N} p(i,k)\log p(i,k),$$

using convention that if $p(i,k) = 0$, then $p(i,k)\log p(i,k) = 0$.

For the purpose of our study of entropy based detection of DDoS attacks we carried out simulations for PSN model setups with network connection topology isomorphic to $\mathcal{L}^{p}_{\square}(37)$ (i.e., periodic square lattice with 37 nodes in the horizontal and vertical directions) and each of the three types of *ecf* (i.e., ONE, QS and $QSPO$). Thus, we considered PSN model setups $\mathcal{L}^{p}_{\square}(37, ecf, \lambda)$, where *ecf* $=$ ONE, or QS, or $QSPO$, and λ is a value of *source load* that the network is operating under normal conditions. We studied DDoS attacks for source load value $\lambda = 0.040$. At this value each PSN model setup is free of any congestion, i.e. is in its free flow state. The critical source load value λ_c , i.e. the phase transition point from free flow to congested network state, for each of the considered PSN model setups is as follows: $\lambda_c = 0.053$ for $\mathcal{L}^{p}_{\square}(37, ONE)$, and $\lambda_c = 0.054$ for $\mathcal{L}^{p}_{\square}(37, QS)$ and $\mathcal{L}^{p}_{\square}(37, QSPO)$. For details how to estimate λ_c see [3]. Since we always start simulations of PSN model setups with empty queues, each time we started DDoS attacks after the initial transient time, i.e., when the network was operating already in its normal steady state for some time. For each of the considered PSN model setups we started the DDoS attacks at time $k_0 = 20480$ that was much larger than the transient times and allowed the attacks to last until the final simulation time, $T = 131072$ (the same for all PSN model setups).

For each PSN model setup operating under normal conditions, i.e., in the absence of any attack and for each considered set M of monitored routers we calculated first entropy function $H(M,k)$. Thus, we built first a "natural" entropy function, a sort of "fingerprint" profile of the given PSN setup, characterizing normal PSN operation, i.e. normal traffic. Next, we calculated entropy function $H(M,k)$ for each PSN model setup being under a DDoS attack. We considered a series of separate DDoS attacks each characterized by a number of active "zombies". In this series, while increasing number of "zombies", we maintained locations of the "zombies" from the previous DDoS attacks, i.e. we added only new "zombies". We calculated entropy functions $H(M,k)$ for sets M of different sizes, i.e. with different numbers of monitored routers. Also, for each number of monitored routers we considered sets M that differed in locations of monitored routers. We selected monitored routers locations randomly using different seeds of random number generator.

As mentioned earlier, for each monitored set of routers our entropy algorithm first builds a "natural entropy" or "fingerprint" profile of the network's normal behaviour in the absence of any attack. For the considered PSN model setups our simulations showed that entropy functions of packet traffic monitored at 5% of all network routers (i.e., at 68 routers out of 1369 routers in our model) may already deviate significantly downward from their "natural profiles" when the network is under DDoS attack with four or more "zombies", see Fig. 1, Fig. 2, and Fig. 4 We observe on these figures that if DDoS attack is sufficiently strong the entropy functions almost immediately and sharply deviate downward from

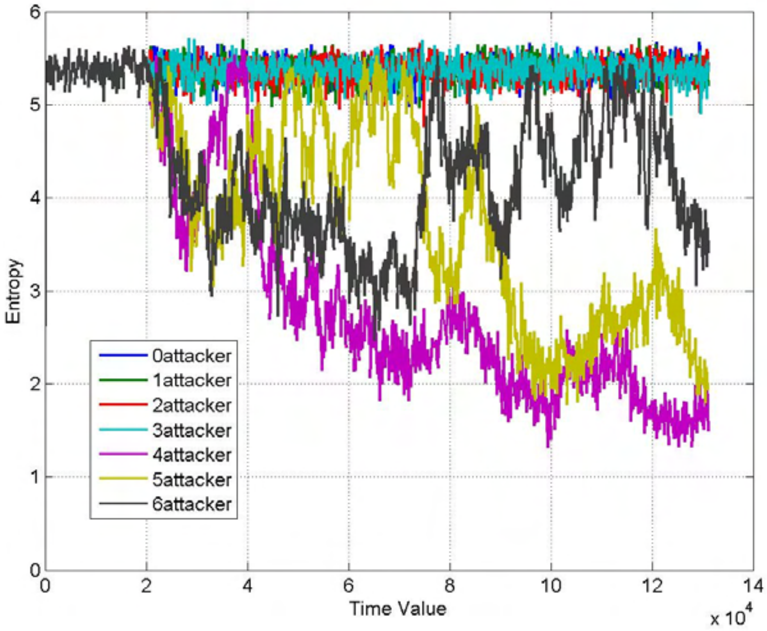


Fig. 1. Plots of “natural entropy” profile (i.e., with 0 attackers) and entropy functions of packet traffic monitored at 5% of randomly (with seed 1) selected routers during DDoS attacks in PSN model with $\mathcal{L}^{p}_{\square}(37, ONE, 0.040)$ setup. The horizontal superimposed plots correspond to attacks with 0, 1, 2, and 3 attackers. The initial decrease in entropy of packet traffic is similar for DDoS attacks with 4, 5 and 6 attackers. However, after the initial transient time the dynamics of entropy functions is different for these attacks.

the ““natural entropy” profiles. In all the figures the horizontal plots correspond to “natural entropy” profiles calculated for the selected sets of monitored routers. The locations of monitored routers in Fig. 1 and Fig.3 are the same but they are different from the locations of the routers in Fig. 2, and Fig. 4 which are the same on both of these figures. In Fig. 1, Fig. 2, and Fig. 4 plots of entropy functions for network under DDoS attack with 1, or 2, or 3 “zombies” are superimposed with the respective entropy “fingerprint” plots, i.e when number of attackers is “0”. This is because for these numbers of “zombies” the attacks are weak, in particular, when networks are using adaptive routings. Our simulations showed that to detect weak DDoS attacks one needs to monitor larger number of routers. In Fig. 3 all entropy plots corresponding to DDoS attacks are superimposed with the “fingerprint” plot. From Fig. 3 and Fig. 4. we see that for a give number of monitored routers behaviours of calculated entropy functions of packet traffics depend on routers locations. Thus, in a case of weak attacks and a small number

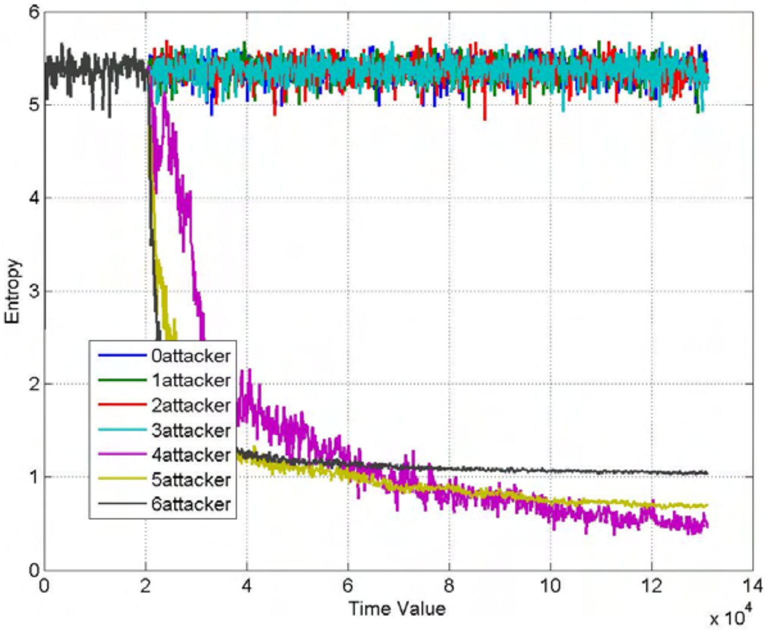


Fig. 2. Plots of "natural entropy" profile (i.e., with 0 attackers) and entropy functions of packet traffic monitored at 5% of randomly (with seed 2) selected routers during DDoS attacks in PSN model with $\mathcal{L}^{p}_{\square}(37, ONE, 0.040)$ setup. The horizontal superimposed plots correspond to attacks with 0, 1, 2, and 3 attackers. The initial decrease in entropy of packet traffic for DDoS attack with 5 and 6 attackers is faster than for the attack with 4 attackers.

of monitored routers the selection of their locations can affect significantly our ability to detect DDoS attacks. Our simulations showed that by adding to the original set additionally 15% of routers out of their total number in a network (i.e, by monitoring 20% of all routers, in our case 274 out of 1369) the entropy functions of packet traffic behave qualitatively the same as in Fig.4. Thus, one can detect anomalies in packet traffic using them because when the values of entropy functions sharply decrease from the "fingerprint" profiles shortly after the beginning of DDoS attacks this means that they detect with certainty the presence of an infrequent event, i.e. an emerging anomaly in packet traffic. In our simulations these anomalies were caused by DDoS attacks.

Our simulations showed that for each set M of monitored routers the plots of entropy functions of packet traffic for PSN model setups with *ecf* QS and $QSPO$ are qualitatively and quantitatively similar. However, they differed from the corresponding entropy functions of packet traffic for PSN model setup with *ecf* ONE. This can be seen from the figures. Fig. 1 and Fig. 2 display entropy

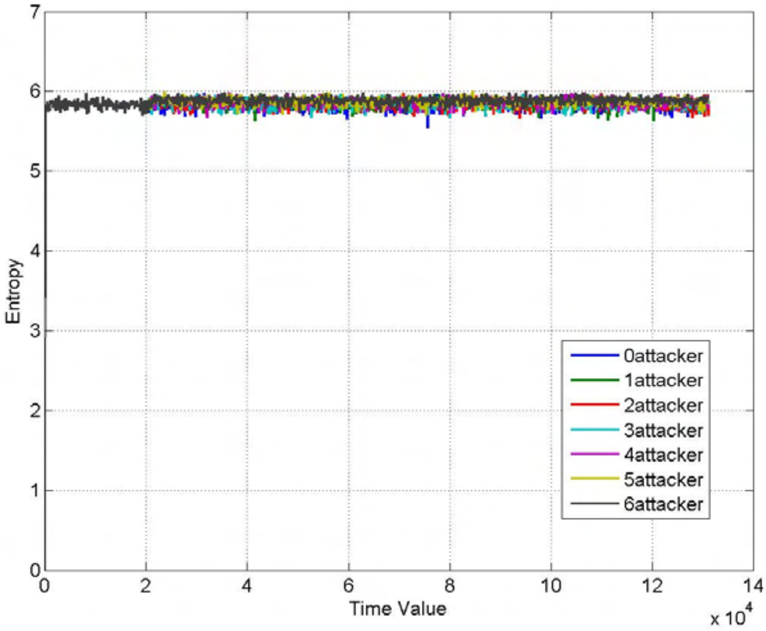


Fig. 3. Plots of “natural entropy” profile (i.e., with 0 attackers) and entropy functions of packet traffic monitored at 5% of randomly (with seed 1) selected routers during DDoS attacks in PSN model with $\mathcal{L}^{p}_{\square}(37, QSPO, 0.040)$ setup. All entropy plots are superimposed.

functions for PSN model setup using *ecf* ONE and Fig. 3 and Fig. 4 display entropy functions for PSN model setup using *ecf* $QSPO$. Thus, our ability to detect DDoS attacks depends also on the type of routing algorithm used by a network. It is much easier to detect DDoS attacks by calculating entropy functions of packet traffic for networks using static routing than dynamic ones. The static routings do not have the ability to route packets avoiding congested network nodes. Thus, congestion develops very quickly along the paths from “zombies” to the victim and around the victim altering “natural packet traffic” and the entropy “fingerprint” profiles. For networks using dynamic routings packet traffic is more evenly distributed across each network and it takes longer for congestion to develop, most likely, first around the victim and from there to spread out into the network. The build up of congestion and spatio-temporal packet traffic dynamics under DDoS attacks we will discuss elsewhere.

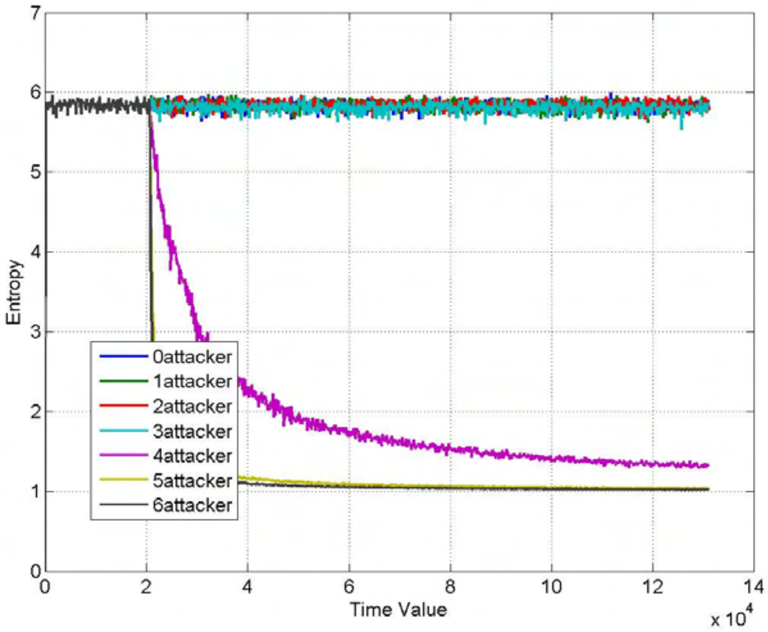


Fig. 4. Plots of "natural entropy" profile (i.e., with 0 attackers) and entropy functions of packet traffic monitored at 5% of randomly (with seed 2) selected routers during DDoS attacks in PSN model with $\mathcal{L}^{p}_{\square}(37, QSPO, 0.040)$ setup. The horizontal superimposed entropy plots correspond to DDoS attacks with 0, 1, 2, and 3 attackers. The entropy decrease for DDoS attack with 4 attackers is smaller than for attacks with 5 and 6 attackers for which the entropy plots are almost superimposed.

7 Conclusions

DDoS attacks change "natural" spatio-tempral packet traffic patterns, i.e. "natural" distributions of packets among routers. We have demonstrated that these changes may be detected by calculating entropies of packet traffic distributions among a small number of selected monitored routers. Thus, one can detect anomalies in packet traffic using entropy based detection methods because the values of entropy of packet traffic sharply decrease from the "fingerprint" profiles shortly after a start of DDoS attack, meaning with certainty presence of an infrequent event, i.e. an emerging anomaly in packet traffic. In our simulations these anomalies were caused by DDoS attacks. Our simulations showed that to detect changes in entropy caused by weak DDoS attacks one needs to monitor larger number of routers and/or pay attention to their locations within the network while selecting them for monitoring. We have observed that stronger DDoS attacks cause significant and almost immediate changes in entropy of packet traffic monitored even at a small number of routers regardless of their position. Additionally, we observed that it is much easier to detect DDoS attacks by calculating entropy of packet traffic for networks using static routing than dynamic ones. In conclusion, we demonstrated the ability of entropy to detect DDoS attacks. However, several questions need to be explored further, i.e. how to select the monitored routers and how many of them so that entropy can detect anomalous packet traffic regardless of its intensity.

8 Acknowledgments

The authors acknowledge the prior work of A.T. Lawniczak (A.T.L.) with A. Gerisch and X. Tang and the use of Sharcnet computational resources. A.T.L. acknowledges partial financial support from Natural Science and Engineering Research Council (NSERC) of Canada and H. Wu from the Univ. of Guelph. B. N. Di Stefano acknowledges full financial support from Nuptek Systems Ltd.

References

[1] A.T. Lawniczak, A. Gerisch, and B. Di Stefano. Development and Performance of Cellular Automaton Model of OSI Network Layer of Packet-Switching Networks, Proc. IEEE CCECE 2003- CCGEI 2003, Montreal, Quebec, Canada (May/mai 2003), pp. 001–004, 2003.

[2] A. Gerisch, A.T. Lawniczak, and B. Di Stefano. Building Blocks of a Simulation Environment of the OSI Network Layer of Packet Switching Networks, Proc. IEEE CCECE 2003-CCGEI 2003, Montreal, Quebec, Canada (May/mai 2003), pp. 001–004, 2003.

[3] A.T. Lawniczak, A. Gerisch, and B. Di Stefano. OSI Network-layer Abstraction: Analysis of Simulation Dynamics and Performance Indicators, Science of Complex Networks, J. F. Mendes, Ed., AIP Conference Proc., vol. 776, pp. 166–200, 2005.

[4] A.T. Lawniczak, A. Gerisch, K.P. Maxie and B. Di Stefano. Netzwerk: Migration of a Packet-Switching Network Simulation Environment from MS Windows PC to Linux PC and to HPC, IEEE Proceedings of "HPCS 2005: The New HPC Culture The 19th International Symposium on High Performance Computing Systems and Applications", Guelph, May 15–18, 2005, pp. 9.

[5] "Paul Baran and the Origins of the Internet" http://www.rand.org/about/history/baran.html

[6] http://en.wikipedia.org/wiki/Phishing

[7] http://en.wikipedia.org/wiki/Computer_worm

[8] http://en.wikipedia.org/wiki/Computer_virus

[9] http://en.wikipedia.org/wiki/Denial-of-service_attack

[10] http://en.wikipedia.org/wiki/ICMP_Echo_Request

[11] http://en.wikipedia.org/wiki/Ping_flood

[12] http://en.wikipedia.org/wiki/Ping_of_death

[13] http://www.theregister.co.uk/2002/10/23/feds_investigating_largest_ever_internet/

[14] http://en.wikipedia.org/wiki/Mafiaboy#cite_note-13

[15] J. Yuan and K. Mills (2005): Monitoring the Macroscopic Effect of DDoS Flooding Attacks, IEEE Transactions on Dependable and Secure Computing, Vol. 2, No. 4, 1–12.

[16] A.Nucci and S. Bannerman (2007): Controlled Chaos, December 2007 IEEE Spectrum, 43–48

On modeling of the heart pacemaker by cellular automata — topology issue

Danuta Makowiec

Institute of Theoretical Physics and Astrophysics, Gdańsk University,
80-952 Gdańsk, ul.Wita Stwosza 57, Poland
fizdm@univ.gda.pl

Abstract. Greenberg-Hastings (GH) cellular automata are known to mimic electrical properties of cardiac cells. The GH rule is modified to resemble self-excitation of the heart pacemaker cells. A plain square lattice connections, usually used in cellular automata models of the cardiac tissue, are rewired to induce heterogeneity in the intercell connections. The rewiring rule is local in relation to cellular automata construction's idea, and the rule sets up the preference to dense connectivity between cells. Simulations show that the heterogeneity in topology qualitatively influences oscillatory properties of the system. The intrinsic dynamics of the cellular automaton drives the self-organization of network state to the oscillatory one. However, the strength and diversity of oscillations depend not only on intrinsic cellular dynamics but also on underlying topology. Large variety in oscillations is interpreted as the better flexibility to response effectively to the actual needs of the body.

1 Introduction

Gil Bub, Alvin Shrier and Leon Glass in [1, 2] have discussed simple two dimensional cellular automata (CA) to explain the role of intercell relations in propagation of impulses in the cardiac tissue. They compared the appearance of multiple spirals on CA states to the electrical activity of the ventricle undergoing the fibrillation. They found that the impaired communication between cells led to the breakup of the spiral waves.

The model considered by Bub *et. al* is the adaptation of the Greenberg-Hastings (GH) model [3, 4]. In the original GH model an automaton a_I is assigned to each site of a regular lattice. The subscript I refers to the location in the lattice. The neighborhood of a site corresponds to its K nearest neighbors. The states of the automaton are *firing* (F), *refractory* (R) and *activity* (A). The update rule is as follows:
— if $a_I(t) = F$ then $a_I(t+1) = R$
— if $a_I(t) = R$ then $a_I(t+1) = A$
— if $a_I(t) = A$ then $a_I(t+1) = A$ unless the number of automata in the *firing* state at time step t in the neighborhood of I is greater than some threshold T_F; then $a_I(t+1) = F$.

The three-state CA model can be easily generalized to the n-state CA model by considering time intervals n_F and/or n_R in which the automaton stays in *firing* and *refractory* states correspondingly, and then switches to the next state.

Bub *et. al* modified the GH model to account for the physiological property of the spontaneous activation of a cardiac cell. They considered the possibility of self-excitation of each automaton:
— if $a_I(t) = A$ then, with probability p, $a_I(t+1) = F$ independently of the states of its neighbors.

Moreover, they also found that the spatial heterogeneity is an important ingredient in the global organization of dynamics in the considered system [2]. Therefore they considered the regular two-dimensional system proposed by Markus and Hess [5] modified by implementing different local cell densities.

Spontaneous activation is the inherent feature of each cardiac cell which constitute the pacemakers [6]. The regular impulses, that result in rhythmic contractions of the heart, begin at the first cardiac pacemaker called the sinoatrial (SA) node. The activity of the SA node spreads throughout the atria causing the atrial contraction. This activity is passed to the atrioventricular (AV) node — the second cardiac pacemaker. Specialized conduction pathways: bundle of His and Purkinje fibers conduct the impulse throughout the ventricles causing the ventricle's contraction in unison.

At AUTOMATA 2007 workshop in Toronto, we presented features of a one-dimensional system of CA where each automaton had the property of self-excitation, namely, each automaton must switch to the *firing* state after n_A steps spent in the *activity* state. We referred to such CA as FRA-CA.

Our studies of the two FRA-CA have shown that only the following stable states are possible, [7, 8]: *the rules adjusted evolution* — if the result of both rules: intrinsic and interactions between the automata, is the same; *the alternating impacts evolution* — if within each period two events of impacts take place: the first event means **A** automaton is impacted by **B** automaton —**A** is switched to *firing*, and then the second event occurs — **B** is impacted by **A** what switches **B** to *firing*; *the quiet evolution* — there are not any impacts between the automata.

One can say that the alternating impacts evolution is the maximally active dynamics since both cells of a pair intensively interact with each other all the time. Because of the intensity of impacts the intrinsic periods of both automata are shortened to the shortest period possible $T^* = n_F + n_R + 1$. The other two solutions: the rules adjusted evolution and the quiet evolution, are also periodic but the period is equal to the intrinsic period of the FRA-CA, i.e., to $T = n_F + n_R + n_A$.

Moreover, it has appeared that in case of a line of FRA-CA with the open boundary, the system always reaches a periodic state with only one of the two periods either T or T^* . Depending on the model parameters the probability to find which of the two periods T or T^* occurs, depends on the relation between n_F and n_R. If $n_F \leq n_R$, then we have only the solution with the period T. If n_F is slightly greater than n_R, then the solution with the period T^* is significantly more probable. Finally, if $n_F >> n_R$, then the solution oscillates with the T^* period .

It has also been found that if $n_R > n_F$ then all automata follow the rules adjusted evolution. It is possible because the rule adjusted evolution is of the diffusive type. This case is physiologically interesting because it is known that the time used by a cardiac cell for the *firing* state is shorter than the time spent in the *refractory* state or during the *activity* state [9]. Therefore in our further analysis, though trying to be general, we concentrate on FRA-CA systems which have this property.

Searching among details of particular configurations of systems which evolve with the period T^* we have discovered that for emerging stabilization with T^*, it is enough that there exists a single pair of automata which performs the alternating impacts evolution. The periodicity of that pair is then propagated to the both ends of a line because the rest of the pairs have their phases adjusted. Hence, high frequency oscillation of the whole system results from self-organization of all automata and, moreover, it relies on the two automata activity. In this sense the line of FRA-CA can be considered as the approximation to the cardiac pacemaker. However the arrangement in a line is far from the known cellular organization of the SA node. Therefore it is interesting to investigate how other network relations influence the FRA-CA system properties as the model of the cardiac pacemaker.

In the following, we concentrate on the influence of the network topology to generate regular oscillations of the total CA state. The FRA-CA will be located in nodes of the complex network, however, the network design will be strongly physiologically motivated [10, 11]. Section 2 contains both the physiological motivation and details of the network construction. Then, in Section 3, we will investigate the periodicity in the FRA-CA systems. In particular we will consider the problem whether the sharp transition between the two periods T and T^* is still present. We especially concentrate on the systems where $n_R = n_A = 2n_F$. The paper is concluded in Section 4.

2 The network topology

Until the middle of the previous century the heart tissue was considered as a syncytium — a multi nucleated mass of cytoplasm that is not separated into individual cells [10]. Due to the development of the electron microscopy it became clear that cardiac cells — myocytes, were long but individual units, bounded on ends by intercalated discs. Soon it was found that each disc was a measurable gap which separated the opposing cell membranes. That gap junction has appeared as highly structured. Each gap junction consists of many mechanisms which provide a pathway for direct cell-to-cell communication between adjacent ones. Therefore in a simplified picture one can approximate the cardiac tissue by a network consisting of branched chains of elongated cells which are connected by gap junctions — the only way to transmit the interactions.

Some network characteristics of the cardiac tissue are known. It appears that a typical myocyte which constitutes the canine SA node has about 4.8 ± 0.7 nearest neighbors which are located as [11]: (a) about 0.7 ± 0.5 side-to-side, (b) about 0.6 ± 0.7 end-to-end ,and (c) about 3.5 ± 1.9 lateral.

Hence about 74% of connections are lateral. Moreover, the side-to-side and lateral connections have relatively small gap junctions, and therefore their efficiency in transmiting signals is considered less effective than in the case of the end-to-end connections. Notably the crista terminalis cells (the cardiac tissue which conducts signals from the SA node to the AV node) have about 60% of 6.4 ± 1.7 neighbors connected as end-to-end.

Taking into account the physiological observations we see that the linear topology is evidently to simple to represent the connections between SA nodal cells. The next topological candidate is a square lattice. Let us consider the regular square lattice with the linear size L. To replicate the nodal properties listed above we introduce the preference to lateral connections in the following way:

(I) For a given probability d, a vertical or horizontal link is created with $d/2$ probability while any diagonal edge is created with $2 * d$ probability.

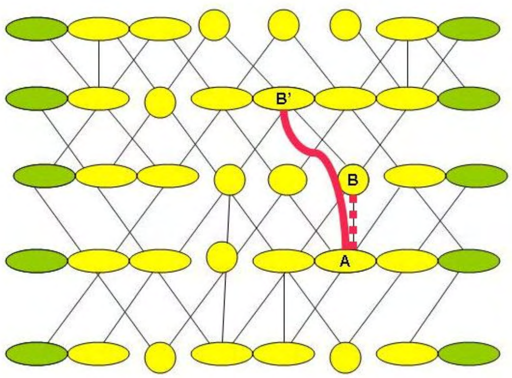


Fig. 1. Construction of a network: most of connections are diagonal because of **(I)** rule; the leftmost and rightmost cells of each row are outputs of the system (inputs to crista terminalis) and all of them are linked to their neighbors by the end-to-end connections; the red lines illustrate the rewiring rule **(II)**: the **AB** edge is exchanged by the **AB'** edge. Color on-line.

It is easy to see that the canine SA node structure is about to be restored if we work with $d = 0.45$, see Fig. refnetwork. The cells from the leftmost column and rightmost column are considered as the interface to the *c*rista terminalis cells. Therefore we additionally link them via horizontal connections to its neighbors. Moreover, it happens, especially when d is small, that isolated cells appear. To cure this unrealistic situation, we connect such cells to their nearest right cells.

By these two extra rules, some additional horizontal connections are present in the system. In case of $d = 0.45$ the resulting network has about 10% vertical, 11% horizontal and 79% of diagonal connections.

The introduced structure is flat. To make the surface uneven we propose rewiring procedure. The rewiring rule is local to obtain the best possible relation to CA idea, and additionally the rule sets up preference to dense connectivity between a cell and the network. The rewiring consists of the following rules: see Fig. 1:

(II) Let p be the probability of rewiring

(i) For a given cell A when choosing its neighbor to disconnect, a less connected cell is preferred. The probability to unlink the B cell from the vertex A is calculated as follows:

$$p_{unlink} = \frac{p}{\deg(B)}$$

(ii) The rewiring is local what means that a new cell B' will be linked to the cell A chosen only from the actual neighbors of B automaton.

(iii) To preserve the line structure, any horizontal connection is forbidden to be rewired. Unlinking from a leaf is forbidden also.

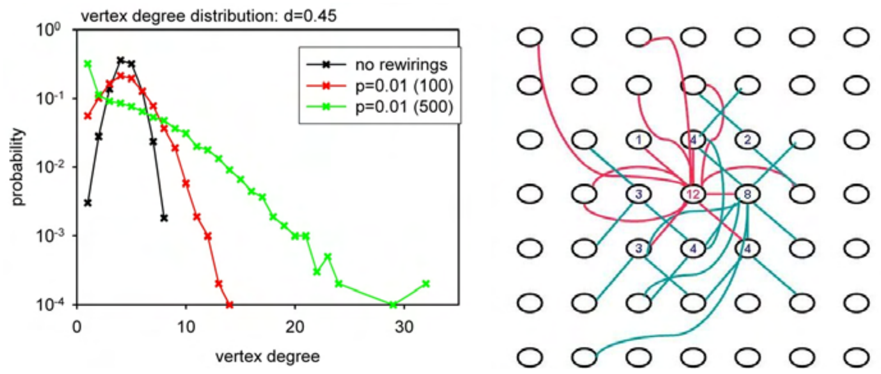


Fig. 2. Left figure: the vertex distribution in considered networks. Right figure: a part of a typical configuration with actual connections. All neighbors of some randomly chosen vertex with high degree are presented — red lines. The green lines correspond to connections within the Moore neighborhood of the chosen vertex. Numbers are degrees of vertices. Color on-line.

In Fig. 2 (left) we show the vertex degree distributions in the networks which result after applying the above algorithm to each edge with $p = 0.01$ and repeating the procedure 0, 100 and 500 Monte Carlo steps (MCS). It appears that due to locality in rewiring the network is only slightly modified, see Fig. 2 (right) for a typical example of the connections which are established after 100 MCS. The network is almost flat but slightly heterogeneous — there are several vertices with the vertex degree twice larger than the average vertex degree.

3 Results

The physiological observations show that the lengths of the *firing* and *refractory* phases of a cell activity are fixed. Hence fixed timings n_F, n_R and n_A are the proper approximation to the reality. However, to weaken the stiffness of deterministic rule let us consider the possibility to shorten values of n_F, n_R and n_A in each time step. Formally, let us propose that a cellular automaton performs the stochastic evolution governed by the following rule:

— If at a time step t the state of the FRA-CA is $\binom{\sigma}{s}(t)$ then in the next time step $t+1$, the state of the isolated FRA-CA is given as:

$$\binom{\sigma}{s}(t) \overset{t\to t+1}{\longmapsto} \begin{cases} \binom{next(\sigma)}{1} & \text{with} \quad \text{probability} \quad (\frac{s}{n_S})^{\xi} \\ \binom{\sigma}{s+1} & \text{with} \quad \text{probability} \quad 1-(\frac{s}{n_S})^{\xi} \end{cases} \tag{1}$$

where $\sigma \in \{F, R, A\}$, $n_\sigma \in \{n_F, n_R, n_A\}$ and $next(F) = R$, $next(R) = A$, $next(A) = F$.

— If a network of N FRA-CA is given and $\binom{\sigma}{s}_I$ denotes the state of the automaton located in the I-th node, and N_I is the set of the Ith node neighbors. Then if $\binom{\sigma}{s}_I(t) = \binom{A}{a}$ and there are more than T_F neighbors $J \in N_i$ such that $\binom{\sigma}{s}_J(t) = \binom{F}{f}$ then $\binom{\sigma}{s}_I(t+1) = \binom{F}{1}$; T_F is the threshold for an external signal to initiate the *firing* state of the Ith cell.

By the above rule there is a nonzero probability to switch the automaton state from the current one the next state in the automaton's intrinsic cycle. If $\xi >> 1$ then we restore the deterministic evolution. Notice, for $\xi > 1$, only very few last steps could be skipped. Therefore the effective timings are closely determined by the values of n_F, n_R and n_A and the basic oscillations have periods are only slightly smaller than T or T^*.

Finally, let as assume that the threshold T_F for firing an automaton equals to 1, i.e., at least two neighbors in the *firing* state are needed to switch an automaton from the state of *activity* to *firing*. However, since the horizontal connections are known to be much larger and more efficient than others, we additionally assume that their influence is doubled. Hence only one left or right neighbor being in the *firing* state activates the adjacent cells.

The system activity must rely on the cell-to-cell connection. Therefore we will observe properties of single cells. Let us investigate properties of (a) an automaton from the output AP_{output}, i.e. a cell chosen from the leftmost or rightmost column, (b) an automaton chosen from a subset of all densely connected automata AP_{leader}, and (c) a typical automaton AP_{cell}. All automata to study are selected at random. Then we convert the states of the chosen automata into the function of the membrane electrical potential — called Action Potential and denoted AP: (the linear approximation is physiologically justified [6]) if the automaton state is $\begin{pmatrix} A \\ a \end{pmatrix}$, then $AP(t) = AP^0_A + a(AP^0_F - AP^0_A)/n_A$; if the state is $\begin{pmatrix} F \\ f \end{pmatrix}$ then $AP(t) = AP^0_F + f(AP^0_R - AP^0_F)/n_F$; finally, for $\begin{pmatrix} R \\ r \end{pmatrix}$ we have $AP(t) = AP^0_R + r(AP^0_F - AP^0_R)/n_R$ where $AP^0_A = -65mV$, $AP^0_F = -40mV$, and $AP^0_R = 20mV$ are the SA nodal cell's Action Potential values at the beginning of the corresponding phases.

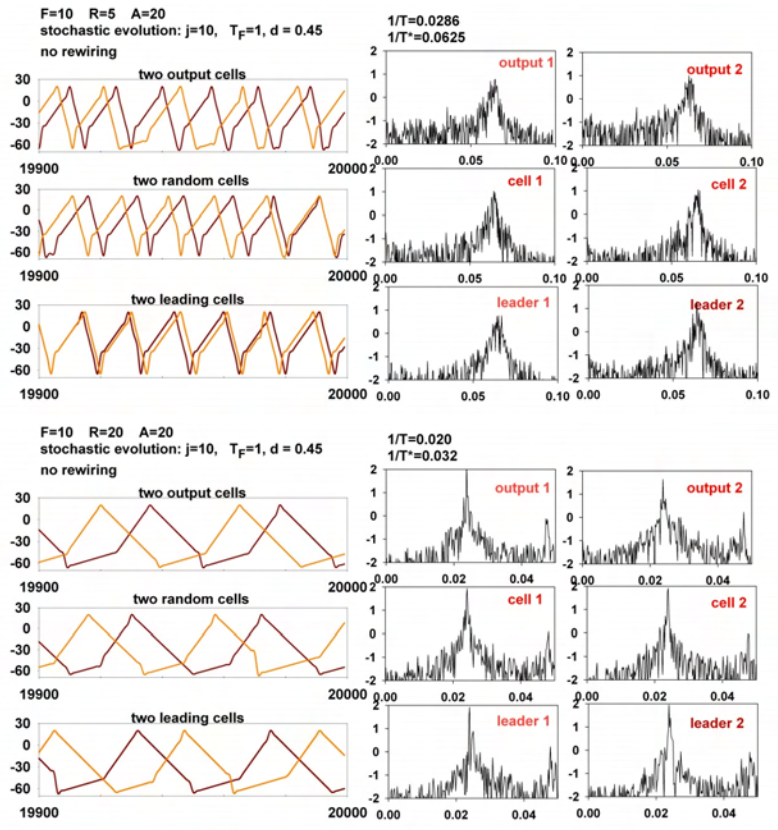


Fig. 3. Left column: typical series of electric activity of randomly selected automata: two of the output cells, two of the whole system, two of the most densely connected. Right column: power spectra of the chosen automata signals, log plots. Upper part corresponds to $n_F >> n_R$, bottom part corresponds to $n_F << n_R$. The rewiring algorithm is not applied.

In Figs. 3–5 we present signals obtained for $n_F = 10$, and for $n_R = 5$ (what refers to the case when n_F is significantly larger than n_R) and $n_R = 20$ (which

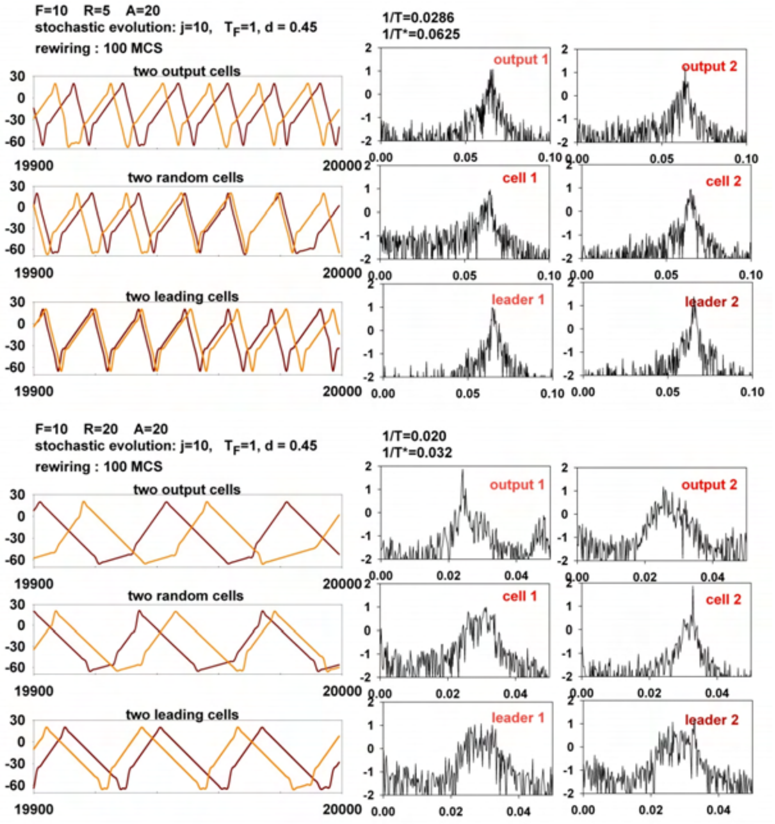


Fig. 4. Left column: typical series of electric activity of randomly selected automata: two of the output cells, two of the whole system, two of the most densely connected. Right column: power spectra of the chosen automata signals, log plots. Upper part corresponds to $n_F >> n_R$, bottom part corresponds to $n_F << n_R$. The rewiring algorithm is applied for 100MCS.

corresponds refers to n_F is significantly smaller than n_R). In these figures we look for the presence of the transition in the dominating oscillation. The value n_A is chosen large, namely $n_A = 20$, to better visualize the difference between the two periods T and T^*. Each figure of 3–5 corresponds to the different network topology. To identify oscillations in signals we calculate the power spectrum $S(f)$ from the 10 000 time steps of stationary signals. The power spectra are found by the Fourier transform.

One should notice that in all plots the maxima in the power spectra are evidently present, though they are wide and moved to the right from the limit values $1/T$ and $1/T^*$. Both these effects are due to the stochasticity in the dynamics.

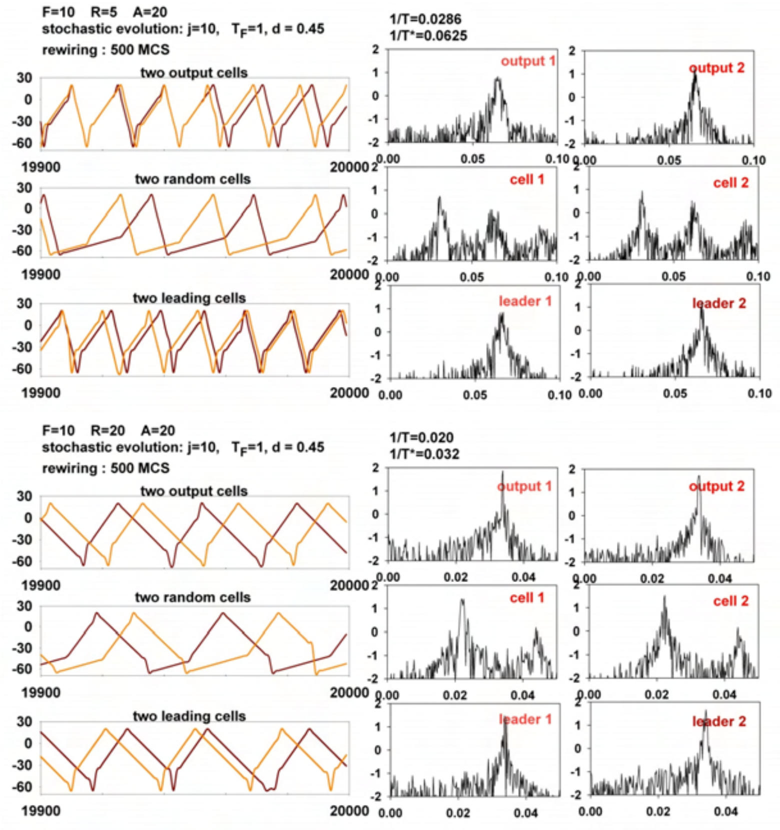


Fig. 5. Left column: typical series of electric activity of randomly selected automata: two of the output cells, two of the whole system, two of the most densely connected. Right column: power spectra of the chosen automata signals, log plots. Upper part corresponds to $n_F >> n_R$, bottom part corresponds to $n_F << n_R$. The rewiring algorithm is applied for 500MCS.

In Fig. 3 — the case when the rewiring algorithm is not applied, we see that there is the transition in the oscillation frequency. If $n_R = 5$ then the basic frequency is closely related to T^* — all randomly selected automata evolve with the shortest possible period, while if $n_R = 20$ then the chosen FRA-CA oscillate evidently with $1/T$.

If the rewiring algorithm is applied for 100 Monte Carlo steps, see Fig. 4, then the dominant frequency for $n_R = 5$ corresponds to T^* as in the case of the flat network. However, the power spectra of signals received from FRA-CA characterized by $n_R = 20$ are significantly different from the ones obtained in case of the flat network. In the plots of the time series, left column of Fig. 4, different lengths of the *activity* phase occur. This directly shows that all frequencies between $1/T$ and $1/T^*$ can be present in the system. Hence the transition here means admitting evolution with the wide spectrum of possible oscillations.

When the structure of the network is modified strongly — the case when the rewiring algorithm is applied for 500 MCS, see Fig. 5, the transition disappears. The dominant frequency is related to the active oscillations with the period T^* independently of n_R, though the oscillations with T are also present. The two randomly selected automata evolve with switching between these two oscillations.

In Figs. 3–5 , $AP(t)$ of few randomly chosen cells are shown. However we also have investigated some average properties of the system. Namely, we have studied oscillations in signals representing the number of FRA-CA staying in the *firing* state in the leftmost column and in the rightmost column, i.e., in the mean output signals. We also have counted the total signal — number of all cells which are in the *firing* state. It has appeared that these signals have the same oscillatory properties as it has been described considering properties of AP of the selected automata. Therefore we can claim that our observations are general.

Now, let us assume that a given frequency f is present in a signal if its power spectrum value $S(f)$ is greater than 1. For the spectra presented in Figs 3–5 in the right columns it means that all fs above zero are collected. In Fig. 6 we present all frequencies extracted in this way from the spectra of AP_{output}, AP_{leader} and AP_{cell} when $n_F = 10$, $n_A = 20$ and for different values of n_R. The results are presented according to the three types of network settings: no rewiring, 100 MCS of rewiring and 500 MCS of rewiring. The black points correspond to $1/T$ and $1/T^*$ frequencies.

Since we have simulated many FRA-CA systems with different values of n_F and n_A, we can state that Fig. 6 is typical with respect to both fixed values. The transition in periodicity is evidently present in all FRA-CA systems located on stochastic homogeneous networks, see Fig. 6 first row. The transition points are about $n_R \approx n_F/2$. Hence, the transition takes place at lower n_R value than it has been observed in the network with the plain line topology. The strength of the oscillations can be estimated by appearance of higher harmonics of the basic frequencies. In all plots obtained for the not rewired network, the second harmonics of the T period is present. So the system basically evolves according to the cellular intrinsic dynamics.

On the other hand, when the network is strongly modified in the result of rewiring algorithm applied for 500 MCS (the bottom row in Fig. 6), the prevailing frequency is related to $1/T^*$. However $1/T$ frequency is also present. Especially cells which are not densely linked to the network may evolve with their intrinsic period.

The case of the network, where the local rewiring rule was applied for 100 MCS, is different from the both cases described above. The wide interval of frequencies is noticeable. This variety of frequencies is related to all possible shortenings of the activity time n_A. It seems that the dynamical self-organization of FRA-CA is not as fixed as it happens when FRA-CA are placed on other considered networks.

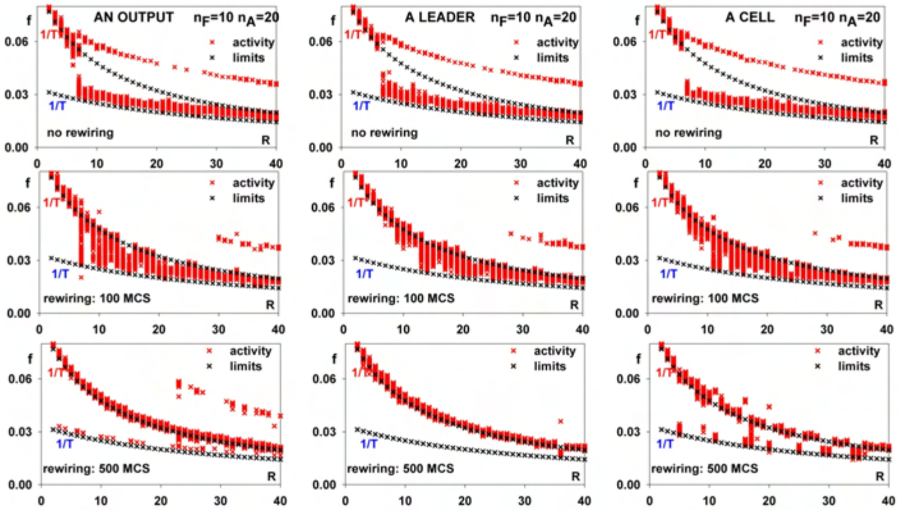


Fig. 6. Oscillations that are strongly present in the power spectra: $\{f : S(f) \geq 1\}$, of cellular signals: AP_{output}, AP_{leader} and AP_{cell} (in columns), for different network models (in rows). $n_F = 10$, $n_A = 20$ and $n_R = 2, 3, \ldots 40$.

In Fig. 7 we show snapshots from a stationary state of the best, in our opinion, FRA-CA pacemaker. One can observe that the activity of the state is governed by clusters of CA which are in the *firing* state. Moreover, these clusters have the spiral shapes. We believe that emergence of such patterns is the indicator that the evolution relies on automata with adjusted phases. We also believe that in the centers of the spiral patterns there are few automata which are tightly joined together by a kind of alternating impacts evolution. These sources of the spiral patterns are long living structures, though a kind of a stochastic walk of these centers can be observed. This walk is probably related to the stochasticity in the intrinsic dynamical rule.

4 Conclusions

Electrophysiological properties of the sinoatrial node have been studied extensively in various species such as rabbits [13] or dogs[11]. But the observations found in these studies are valid also for the human sinoatrial node. Emanuel Drouin observed that “the electrical behavior of adult human SA node pacemaker cells resembles those of SA nodal tissue of different animal species" [12]. He concluded his research on the human SA nodal cells saying “the latter ideally mimic the former". Therefore, when modeling the SA node, one can base on biological observations obtained from investigations of either animals or humans, equivalently.

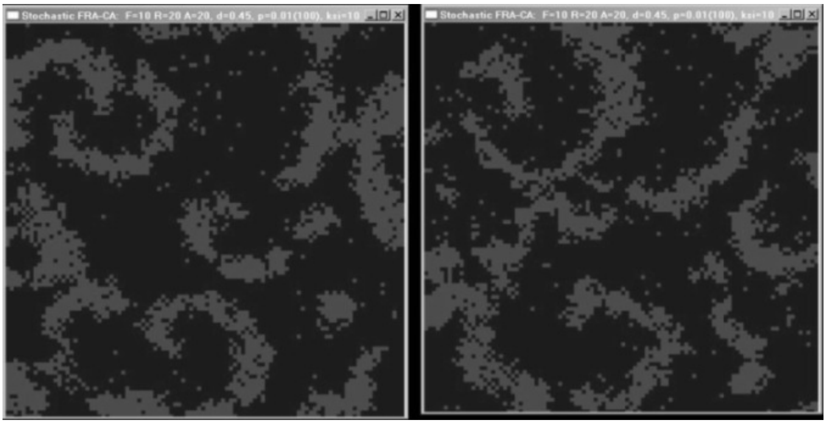

Stochastic FRA-CA: F=10 R=20 A=20, d=0.45, p=0.01(100), ksi=10
Stochastic FRA-CA: F=10 R=20 A=20, d=0.45, p=0.01(100), ksi=10

Fig. 7. Two snapshots from a stationary state of the FRA-CA system on the rewired network. (100MCS, $n_F = 10, n_R = 20, n_A = 20$.) Cells in the *firing* state are plotted in red (gray), other states are plotted in blue (dark gray).

Cellular automata have been used to model biological systems of different types, see, e.g., [14] for the review. In particular, cellular automata are known to model accurately the excitable media, see e.g. [15, 1, 2]. Our proposition is related to the GH model of the excitable media. The GH rule has been adjusted to resemble the self-excitation property of the SA nodal cells. Moreover, the square lattice topology, usually used in cellular automata models of the cardiac tissue, has been modified to mimic the physiologically known fact, namely heterogeneity.

Our simulations have shown that the heterogeneity in topology qualitatively and quantitatively influences on the oscillatory properties of the system. The total state of FRA-CA is self-organized to produce an oscillatory state. However, the strength and the rate of the oscillations depend not only on the properties of intrinsic cellular dynamics but also on the underlying topology. If the automata interconnections are flat (though resembling the vertex degree characterization of the SA nodal connections) then the transition between the two limit frequencies $1/T^*$ and $1/T$ occurs when n_R increases. When the two-dimensional topology is locally modified then we observe the presence of the interval $(1/T, 1/T^*)$ of oscillations. However, if the network connections are strongly modified then only the two limit oscillations $1/T$ and $1/T^*$ are simultaneously present independently of the intrinsic automata dynamics.

Cardiological control means the adjustment of the heart rate. This is achieved by the two contrary acting neuronal systems: parasympathetic and sympathetic. Both systems influence the heart rate by sending bursts of impulses through the neuronal network what leads to release of acetylcholine — the parasympathetic transmitter, or noradrenaline — the sympathetic transmitter, at the myocytes. In response the myocyte elongates (the parasympathetic activity) or shortens (the sympathetic activity) its time to the self-activation [16].

Such control can be directly implemented into the FRA-CA dynamics by elongating or shortening the activity time n_A. However the change will be effective in FRA-CA only if the intrinsic periodicity T is present. In the systems considered the frequency $1/T$ is present however its role is different — from the only possible (the flat network) to staying apart (the most rewired network). Therefore, in our opinion, the FRA-CA systems, where the network of flat intercellular connections is only slightly modified, seem to be better prepared to respond to the external control. Verification of this hypothesis is the aim of our further development of the model.

The simulations were performed with 10 000 FRA-CA. It is known that the human SA node consists of about 70 000 cells[6]. Hence to obtain the one-to-one mapping we should increase the size of simulated systems. Moreover, it is known that the SA nodal cells are not identical. Fortunately, the differences between cells are systematic — the further from the center of the sinus node, the difference between a center cell and a periphery cell is more evident. Therefore then enlarging the system one can easily incorporate the fact of the cell diversity.

Acknowledgments. This work is supported by University of Gdańsk, BW-5400-5-0090-8.

References

[1] Bub, G., Shrier, A., and Glass, L.: Spiral wave generation in heteroneous excitable mediaeous excitable media. Phys.Rev.Lett. 88, 058101-1–058101-4 (2002)

[2] Bub, G., Shrier, A., and Glass, L.: Global organization of dynamics in oscillatory heterogeneous excitable media. Phys. Rev. Lett. 94, 028105-1–028105-4 (2005)

[3] Greenberg, J.M., and Hastings, S.P.: Spatial patterns for discrete models of diffusion in editable media. SIAM J. Appl. Math. 34, 515–523 (1978).

[4] Greenberg, J.M., Greene, C., and Hastings, S. P.: Remarks on a 25 year old theorem on two-dimensional cellular automata. International Journal of Unconventional Computing, 1 399–402 (2005)

[5] Markus, M., and Hess, B.: Isotropic cellular automaton for modeling excital media. Nature 347, 56–58 (1990)

[6] Klabunde, R.E.: Cardiovascular physiology concepts. on *http:// www.cvphysiology.com* (2007)

[7] Makowiec, D.: On cellular automata modeling of cardiac pacemaker. to appear in *Journal of Cellural Automata* (2008)

[8] Makowiec, D.: Cellular automata model of cardiac pacemaker. to appear in *Acta Physica Polonica B* (2008)

[9] Kodama, I., Honyo, H., and Boyett, M.R.: Are We Lost in the Labyrinth of the Sinoatrial Node Pacemaker Mechanism? Journal of Cardiovascular Electrophysiology 13 (12), 1303ñ1305 (2002)

[10] Saffitz, J.E., Lerner, D.L., and Yamada,K.A.: Gap junctions distribution and regulation in the heart. in *Cardiac Elecrophysiology. From Cell to Bedside*, D.P.Zipes, J.Jalive (Eds.) pp. 181–191. Saunders Co., Philadelphia, PA USA, 2004

[11] Luke, R.A., and Saffitz, J.E.: Remodeling of ventricular conduction pathways in healed canine infarct border zones. J. Clin. Invest. 87(5), 1594ñ1602 (1991)

[12] Drouin, E.: Electrophysiologic properties of the adult human sinus node. Journal of Cardiovascular Electrophysiology 8 (3), 254-ñ258 (1997)

[13] Kodama, I., Honyo, H., Dobrzynski, H., and Boyett, M.R.: Cellular mechanisms of sinoatrial activity. in *Cardiac Elecrophysiology. From Cell to Bedside*, D. P. Zipes, J. Jalive (Eds.), pp 181–191. Saunders Co., Philadelphia, PA USA, 2004

[14] Ermentrout, G.B., and Edelstein-Keshet, L.: Cellular automata approaches to biological modeling. J. Theor. Biol. 160, 97–133 (1993)

[15] Saxberg, B.E.H., and Cohen,R.J.: Cellular automata models of cardiac conduction. In *Theory of Heart: Biomechanics, Biophysics,, and Nonlinear Dynamics of Cardiac Functions* L.Glass, P.Hunterans and A.McCulloch (eds.) Springer-Verlag, New York, 1991.

[16] DiFrancesco, D.: I_f inhibition: a novel mechanism of action. European Heart Journal Supplements 5(Supplement G), G19–G25 (2003)

.

A cellular automata simulation of calcium driven tissue differentiation in human skin equivalent models

Graeme J. Pettet[1], Colin P. Please[2], Ricardo L.Colasanti[1], Rebecca A. Dawson[3], and Jos Malda[3,4]

[1] School of Mathematical Sciences, Tissue Repair and Regeneration Program, Institute of Health and Biomedical Innovation, Queensland University of Technology, 2 George St. 4000 Brisbane Qld, Australia.

[2] School of Mathematics, University of Southampton, Southampton SO17 1BJ, United Kingdom.

[3] School of Life Sciences, Tissue Repair and Regeneration Program, Institute of Health and Biomedical Innovation, Queensland University of Technology, 2 George St. 4000 Brisbane Qld, Australia.

[4] Department of Orthopaedics, University Medical Center Utrecht, Room G05.228, PO Box 85500, 3508 GA Utrecht, The Netherlands.

Abstract. Wound healing is a complex physiological process, requiring restoration of the barrier function of skin. Human skin equivalent models (HSEs) have been proposed as a valuable *in vitro* tool for understanding the biology of epidermal wound repair while also providing a significant new platform for studying growth, response and repair of human skin subjected to treatment under strictly regulated conditions. Experimental evidence of regeneration of the epidermis in HSEs demonstrates interesting dynamics as the barrier function is restored. We present a cellular automata model that describes the development of a multilayered structure in order to explore the role for calcium dependent differentiation. Comparison of the simulation model with experimental data shows that the calcium-tissues interaction is an autopoiesis mechanism that sustains the tissue structure.

1 Introduction

Skin, the largest organ in the body, performs many important roles including protection against physical and chemical insults, regulation of temperature and fluid loss. To restore the protective barrier function of skin, the body needs to close any wound damage as quickly as possible. If we are to understand the complex process of wound healing, the employment of biological models is essential. In particular, the use of *in vitro* three-dimensional skin models — human skin equivalents (HSEs) — holds promise for the study of wound healing with and without potential therapeutic agents. Human skin may be viewed as consisting of two parts: a dermis, which is mainly a connected collagen matrix populated by a variety of cellular components including an embedded vascular structure, and an epidermis, which is composed predominantly of a cell type known as keratinocytes in different states of differentiation.

An HSE, as simplified analogue of human skin, is a construct grown upon de-epithelialized and de-cellularised dermis harvested from patients undergoing cosmetic surgery. A central portion of the upper surface of this substrate is seeded with keratinocytes that have been isolated from the same donor [2]. The construct is then maintained in a nutrient bath with the construct positioned at the air-liquid interface. Hence in a fashion analogous to *in vivo* skin, nutrient is supplied to the cells of the epidermis from below, while the upper surface is exposed to the atmosphere. The initial colony of keratinocytes increases in number and spreads laterally, while developing in thickness it quickly establishes a spatial structure analogous with that of human epidermis.

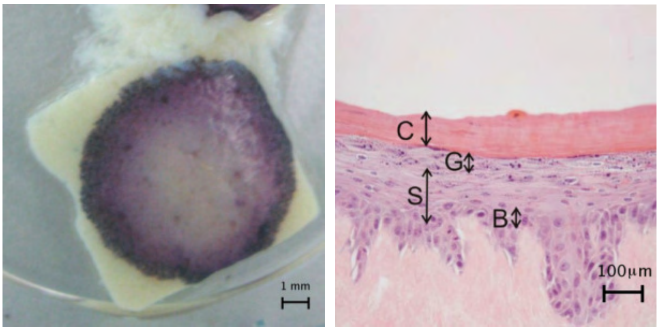


Fig. 1. (left) HSE construct as seen from above. Diameter of colony (stained) approx. 10mm. (right) Histological section through HSE construct showing stratum basale (B), stratum spinosum (S), stratum granulosum (G) and stratum corneum (C). (source: J. Malda)

The epidermis in both the construct and real skin, consists of: stratum basale, stratum spinosum, stratum granulosum overlain by a layer of dead cells known as stratified corneum. Each layer performs a different function and is characterized by keratinocytes in different states of differentiation. The lowermost layer, the stratum basale, contains the only cells that proliferate, usually identified as either keratinocyte stem cells and transit amplifying cells. These proliferating cells are all attached to a basement membrane that separates the dermis from the epidermis. Transit amplifying cells that no longer proliferate detach themselves from the basement membrane and start to differentiate into non-proliferating spinous cells. Continuous cell proliferation at the base of the structure causes cells to be pushed upwards away from the dermis-epidermis interface. As the spinous cell are pushed further from the basement membrane they undergo further differentiation to form the thin stratum granulosum layer. The death of these cells produces the keratinized layer or stratified corneum.

A proposed mechanism modulating the differentiation of the keratinocytes and thus regulating the layering of epithelial tissue depends on the establishment and maintenance of a calcium gradient. Such a gradient is observed in both human skin and the HSE. Calcium has been implicated in the production of strong inter-cell linkages that are most prevalent in the upper stratum granulosum layer and as a trigger for keratinocyte differentiation. The exact origin of this gradient, the manner in which it is maintained is unknown. The elucidation of this mechanism forms the basis for the investigation described in this paper.

Our hypothesis is that the calcium gradient and the differentiated tissue are an autopoiesis system with the calcium gradient maintained by the differentiating tissue while the continued differentiation of keratinocytes and hence the layered structure of the epidermis is driven by the calcium gradient. We have employed a cellular automata (CA) method of simulating this mechanism both because of the discrete nature of the skin cell states and because of CA's proven ability to capture other autopoiesis systems [1, 5, 3].

2 The simulation

In order to prevent any confusion that may arise from our simulation of biological tissue and cells with typical cellular automaton terminology, the individual matrix units of the CA model described in this paper will be referred to as boxes rather than cells. The term cell will be exclusively used for biological cells.

The aim of the simulation is to investigate the relationship between calcium concentration and the spatial pattern of cell differentiation found in the HSE. The simulation is built upon the following simplified biological assumptions about the HSE:

- The HSE epidermal tissue consists of cells and a surrounding extra-cellular fluid.
- Cell proliferation only occurs in a single layer of cells at the base of the epidermis.
- Cell proliferation occurs at a constant rate independent of calcium concentration.
- Any new cell that is produced by proliferation displaces the parent cell from the basal layer.
- Any new cell contains no accumulated calcium.
- Cell proliferation results in adjacent cells being moved within the tissue.
- The calcium concentration at the base of the epidermis is kept at a constant concentration.
- Calcium can diffuse freely throughout the extra-cellular fluid of the tissue.
- If present in the extra-cellular fluid, calcium is accumulated by cells at a constant rate.
- Cells have a constant death rate.
- Cells can also die when they have accumulated a maximum set value of calcium.

– When cells die they release all of their accumulated calcium into the surrounding extra-cellular fluid.
– Dead cells are removed from the tissue and assumed to occupy no space.

2.1 The program

The program is written in Sun Microsystems Java 6.0 and the source code is available at: `http://gpettet.googlepages.com/hse.java` [5]. A java applet of the simulation, embedded in a web page is available at: `http://gpettet.googlepages.com/CalciumTissueModel.html` [6]. The simulation is based on a two dimensional 100 × 100 array of box objects and represents a 2D vertical section through the HSE. The left and right edges of the matrix are linked. A box can contain a single cell object and an integer value of calcium. A box has two sets of values, time now and time next for cell occupancy and calcium. This allows synchronous updating of cell states. A cell can also contain an integer value of calcium and can be in one of two states; connected to the basement (0), or within tissue mass (1).

2.2 The algorithm

```
A. Create a new cell in a box in the middle of the base vector of the matrix.
B. Set that cell state 0.
C. For each box in the base vector of the matrix:
    1. If the box contains a cell, set the number of calcium unit in the box to
       a constant preset value.
D. For each box in the matrix:
    1. If the box contains a cell or is next to a box that contains a cell randomly
       redistribute any calcium units between itself and its 4 surrounding
       neighbours.
    2. If the box does not contain a cell or is not next to a box that contains a
       cell set the calcium value to zero.
E. For each cell in the simulation:
    1. If a random value is less than the death rate value then kill cell and
       transfer any cell calcium to the box that it was occupying and remove
       cell from box.
    2. If the cell is in a box that contains calcium, take up one unit of calcium
       from the box to the cell.
    3. If the cell contains more than 60 units of calcium then kill the cell and
       transfer any cell calcium to the box that it was occupying and remove
       cell from box.
F. For each cell of the simulation that is in state 0 (Basement cells)
    1. If a random number is less than the growth rate;
```

[5] (note page also at `http://ric.colasanti.googlepages.com/hse.java`)

[6] (note page also at `http://ric.colasanti.googlepages.com/CalciumTissueModel.html`)

```
        i. Try each of the four neighbours of the box that the cell occupies;
            a. shift all cells in that direction.
        ii. If the shift has been successful;
            a. move cell.
            b. set cell to state 1.
            c. create new cell in vacated space and set to state 0.
G. repeat C.
```

3 Experiments

The simulation was first run with a calcium refresh value of 2, and run for 6500 iterations. The average tissue thickness was sampled every 100 iterations. At the end of the simulation the distribution of calcium within cells and boxes was noted by averaging the values across each y vector of the matrix. The experiment was repeated with refresh values of 3 and 4. The average tissue thickness was sampled every 100 iterations for each experiment.

4 Results

Figure 2 shows a time series from a simulation of an HSE tissue model for intervals of 100, 600, 1200, and 3000 iterations. The calcium refresh level for this experiment was set at 2 units. The simulated tissue is based on a two dimensional 100×100 array of boxes and shows a 2D vertical section through the HSE. Set (a) shows the distribution of cells within the matrix. The green colored cells at the base of the matrix indicate that these cells are capable of proliferation. The cells above this layer are the non-proliferating spinosum cells that move up through the tissue. The increasing red color of the cells located higher in the tissue indicates the increased levels of calcium that have been accumulated within those cells.

The time series shows an increase in tissue thickness and a subsequent collapse back to a thinner but constant thickness. Set (b) is the matching display of free calcium within the matrix, this is the calcium that is held in the box not the cell. It can be seen that up to the point of tissue collapse there is very little free calcium in the tissue except at the base of the matrix where it is continuously refreshed and from the burst of calcium from randomly dying cells. At the point of tissue collapse however there is a large concentration of free calcium that spreads as a wave across the top of the matrix. This becomes a standing wave of calcium within the thinner but constant thickness of the tissue.

Figure 3 shows the distribution of calcium within the tissue at 6500 iterations, a point at which the tissue has stabilized to a constant thickness. The calcium profile is measured as the average calcium across the complete width of the simulation. The calcium refresh level for this experiment was set at 2 units. The profile shows the levels of free calcium, cell accumulated calcium and the combined total calcium. It can be seen that the graph reproduces the results seen in the time series displays. It shows that the maximum level of free calcium, 10

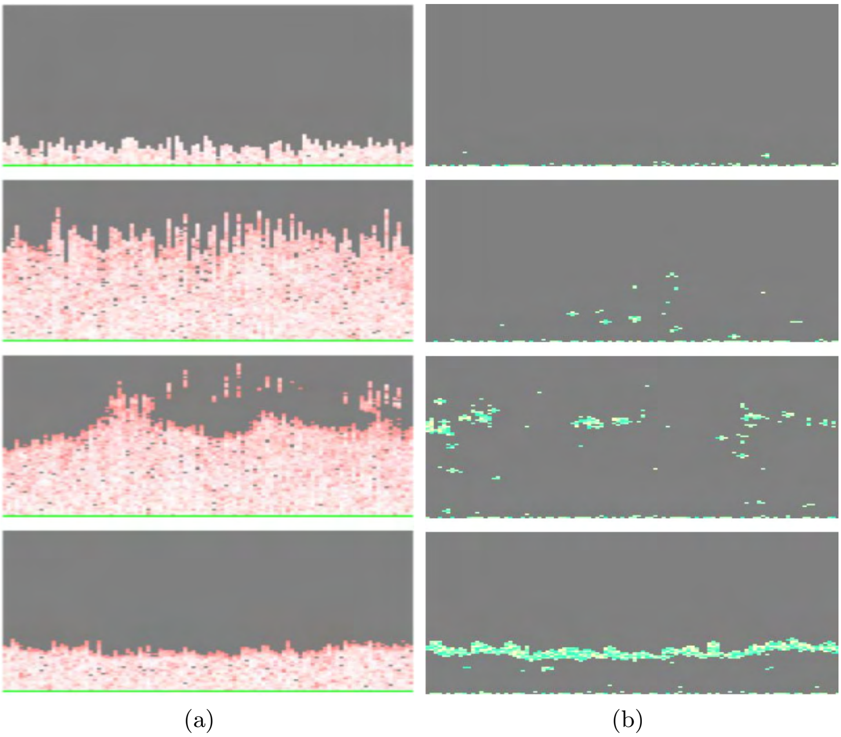


Fig. 2. A time series from a simulation of a human skin equivalent (HSE) model. The simulation is based on a two dimensional 100 × 100 array and represents a 2D vertical section through the HSE. Set (a) shows the distribution of cells, green indicates basement cells while increasing red color represents levels of cell accumulated calcium. Set (b) shows distribution of free or interstitial calcium, increasing blue color represents increasing free calcium levels. The time series is for 100, 600, 1200, and 3000 iterations, increasing down the column. The calcium refresh level was set at 2 units.

units, occurs in a band near the top of the tissue. The concentration falls off either side of this peak. The concentration lower within the cell mass is close to zero only increasing to 2 at the base of the matrix. The level of cell bound calcium increases to a maximum value of 60 units at the top of the tissue and then falls sharply. The total calcium, free plus cell accumulated, follows the same pattern.

Figure 4 shows the evolution of tissue thickness over time for three separate calcium refresh levels: 2,3 and 4 units. All three sets show the same overall behavior that was seen in the time series displays, that the tissue increases in thickness up to a maximum level and then collapses back to a thinner but constant thickness. The graph shows that at the point at which the tissue thickness has stabilized the experiment that has the lowest calcium refresh level, 2 units,

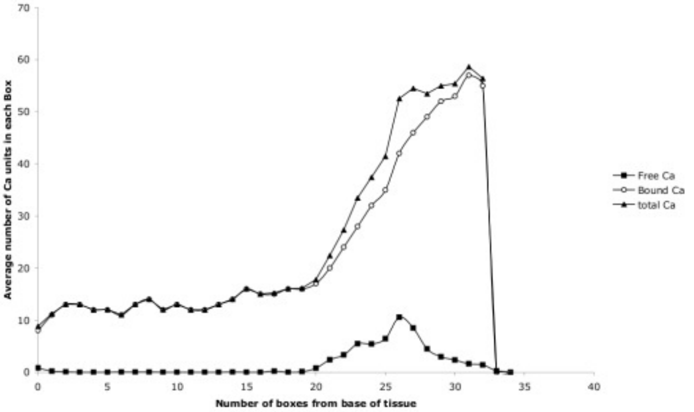


Fig. 3. A measurement of the average calcium distribution within a simulation of an HSE model. The distribution was for a calcium refresh level of 2 units. The profile was taken at 6500 iterations and shows; free calcium, cell accumulated calcium and the combined total calcium.

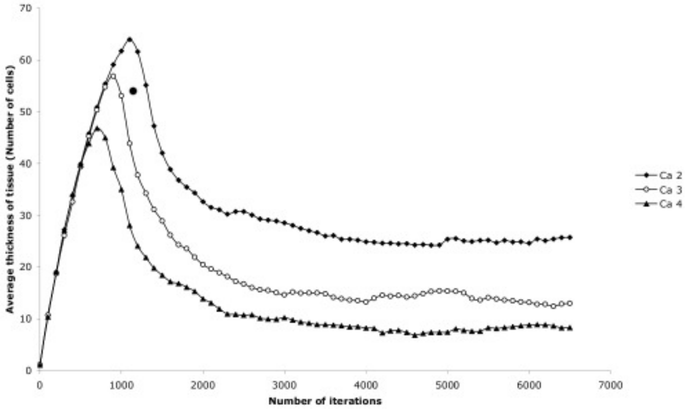


Fig. 4. The change in thickness, measured in average number of cells across the complete array, of a set of simulations of an HSE model for three calcium refresh level of: 2, 3 and 4 units.

has the thickest tissue and the experiment that has the highest calcium refresh level, 4 units, has the thinnest.

5 Discussion

It can be seen from the results that the simulated tissue increases in thickness because of proliferation at the base of the tissue. It can also be seen that while a cell is attached to the base of the tissue it can accumulate calcium from the continually refreshed boxes. This is analogous to the real HSE sitting in a reservoir of nutrient. It can also be seen that when the cell detaches from the base of the tissue it no longer has access to this reservoir of calcium and that the only calcium that it can accumulate must come from freely diffusing calcium within the tissue mass. This calcium can only be derived from calcium expelled from dying cells.

The movement of the cells away from the base through the tissue and their subsequent death creates a transport mechanism for the accumulation of free calcium within the tissue mass. This in turn sets up a feedback mechanism whereby the more free calcium there is in the tissue mass, the higher the likelihood that further cell death and resultant calcium liberation will occur.

Another aspect of this mechanism is that the longer a cell is in the tissue the more likely the cell will take up calcium. Because of the flow of cells through the tissue, the older cells will be found towards the top of the tissue. It will thus be these cells that will saturate first and in turn dump their calcium to their neighbors, in turn causing them to saturate and die. At this point a wave of cell death and calcium liberation occurs. The cells lower in the tissue mass become exposed to the wave of released calcium and they in turn die.

The system is stabilized when the wave front reaches cells within the tissue that will not immediately saturate. This explains why the tissue is thicker when the refresh level of calcium, which is supplied to the base cells, is low. The newly detached cells from the basement will carry with them less calcium and so will need to accumulate more calcium to reach the saturated state. The reason the tissue depth stabilizes is that there is calcium loss from the dying cells at the top of the tissue. This prevents the tissue from filling up with calcium and thus preventing total collapse of the tissue. The structure of the tissue is thus dependent on the calcium profile, but the calcium profile is in turn produced by the differentiating tissue.

This entwined loop is one of the characteristics found in autopoiesis mechanisms [4]. The simulation provides three hypotheses that can be tested with an HSE:

1. That during initial tissue growth there will be very low levels of calcium in the tissue.
2. Calcium will be continuously lost from the HSE at equilibrium.
3. That thicker HSE will be produced with lower calcium.

References

[1] Beer D.R. 2004. Autopoiesis and Cognition in the Game of Life. *Artificial Life* **10**: 309-326.

[2] Dawson, R.A., Upton, Z., Malda, J. and Harkin, D.G. 2006. Preparation of Cultured Skin for Transplantation Using Insulin-like Growth Factor I in Conjunction with Insulin-like Growth Factor Binding Protein 5, Epidermal Growth Factor, and Vitronectin. *Transplantation* **81**: 1668–1676.

[3] Dunkerley, D.L. 1997. Banded vegetation: development under uniform rainfall from a simple cellular automaton model. *Plant Ecology* **129**: 103-111.

[4] Maturana, A. H. R., & Varela, F. J. (1980). Autopoesis and cognition: The realization of the living. Dordrecht, The Netherlands: Reidel.

[5] Wimpenny, J.W.T. and Colasanti R. (1997) A unifying hypothesis for the structure of microbial biofilms based on cellular automaton models *FEMS Microbiology Ecology* 22 (1), 1–16.

Abstracts of invited talks presented at AUTOMATA-2008, Bristol, United Kingdom

1 Nino Boccara (Chicago, USA): Maintenance and disappearance of minority languages – A cellular automaton model

We describe and study a cellular automaton model of immigrants living in a country where the dominant language is different from their own. The cellular automaton represents a neighborhood with a fraction of immigrants who, under the influence of their native neighbors, learn to speak the dominant language. Moreover, these immigrants may also improve their knowledge of the dominant language under the influence of the media (TV, radio, newspapers, etc.), but some immigrants who want to maintain their culture spend some time going to cultural centers promoting their own language. Playing with the different parameters of the model, we study the conditions under which the immigrants' proficiencies in both the dominant and minority languages either increase or decrease.

2 Leon Chua (Berkeley, USA): CA $\in$ CNN

The CNN (Cellular Neural Network) universal chip is a complete programmable supercomputer on a chip. It computes via nonlinear dynamics as flows to attractors in $\mathbf{R}^n$. Applications from gray-scale image processing to brain-like computing will be presented. In the special case where the input image is binary and subject to local dynamics iterating on a `do loop`, the CNN reduces to a cellular automata.

3 Masami Hagiya (Tokyo, Japan): Classification of continuous/discrete state transition systems on space-time

From the viewpoint of cellular automata and dynamical systems, I first classify state transition systems according to three axes: whether states are continuous or discrete, whether space is continuous or discrete, and whether time is continuous or discrete. Among the kinds of state transition systems in the classification, I focus on systems with discrete states, whose space and time are continuous. Those systems are of interest because they are obtained as cells in cellular automata become infinitely small and densely distributed. Transformations from one kind to another in the classification are discussed in some other cases

4 Maurice Margenstern (Metz, France): Cellular automata in hyperbolic spaces: new results

In this talk, we recall the basic tools to implement cellular automata in hyperbolic spaces. We will discuss new results on the localization of cells and novel discoveries on cellular automata in the hyperbolic plane, in particular a variant of Hedlund's theorem for cellular automata in the hyperbolic plane.

5 Tommaso Toffoli (Boston, USA): Lattice-gas *vs* cellular automata: the whole story at last

"I do not know of any single instance where something useful for the work on lattice gases has been borrowed from the cellular automata field. Lattice gases differ in essence from cellular automata. A confusion of the two fields distorts our thinking, hides the special properties of lattice gases, and makes it harder to develop a good intuition." [Michel Henon(1989)].

The political arena of fine-grained parallel computation seems to incite us to take sides for one of two candidates—Cellular Automata (CA) and Lattice-Gas Automata (LG). What is the poor researcher supposed to do? My presentation is intended to be a "Guide to the Perplexed."

Cellular automata provide a quick modeling route to phenomenological aspects of nature—especially the emergence of complex behavior in dissipative systems. But lattice-gas automata are unmatched as a source of fine-grained models of fundamental aspects of physics, especially for expressing the dynamics of conservative systems.

In the above quote, one may well sympathize with Henon's annoyance: it turns out that dynamical behavior that is synthesized with the utmost naturalness when using lattice gases as a "programming language" become perversely hard to express in the cellular automata language. Yet, Henon's are visceral feelings, not argued conclusions. With as much irritation one could retort, "How can lattice gases differ 'in essence' from cellular automata if they are merely a subset of them? What are these CA legacies that may 'distort our thinking' and 'hide the special properties of lattice gases'? And aren't there dynamical systems that are much more naturally and easily modeled as cellular automata?"

Today, with the benefit of twenty years' hindsight—and especially after the results of very recent research—we are in a position to defuse the argument. Henon's appeal could less belligerently be reworded as follows: "Even though CA and LG describe *essentially the same class* of objects, for sound technical and pedagogical reasons it is expedient to deal with them in separate chapters—even separate books for different audiences and applications. What is ox in the stable may well be beef on the table."

The bottom-line message is that these two modeling approaches do not reflect mutually exclusive strategies, but just opposite tradeoffs between the structural complexity of a piece of computing machinery and its thermodynamic efficiency. By casting essential aspects of dynamics in a precise formal context, it becomes possible to explicitly show *why*

– total recycling of information waste can in principle be achieved even in *noninvertible* dynamics;

– at the same time, while waste is so easy to produce if one insists on using simple machinery operating on a local scale, *effective recycling* of information waste may not be possible without *very complex* machinery insuring coordination on a *wide-range* scale. Do we have a case for “Logic for capitalists?" (cf. “Logic for conservatives: The heath-death of a computer," The Economist, 11 Feb 1989.)

Index

Cellular automata are regular uniform networks of locally-connected finite-state machines. They are discrete systems with non-trivial behaviour. Cellular automata are ubiquitous: they are mathematical models of computation and computer models of natural systems.

The book presents results of cutting edge research in cellular-automata framework of digital physics and modelling of spatially extended non-linear systems; massive-parallel computing, language acceptance, and computability; reversibility of computation, graph-theoretic analysis and logic; chaos and undecidability; evolution, learning and cryptography.

The book is unique because it brings together unequalled expertise of inter-disciplinary studies at the edge of mathematics, computer science, engineering, physics and biology.

ISBN 978-1-905986-16-3
90000
9 781905 986163