

LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

AUTOMATIC SURFACES

Bruno Martin, Christophe Papazian

Projet RECIF

Rapport de recherche
ISRN I3S/RR-2007-20-FR

Juillet 2007

Automatic surfaces

Bruno Martin and Christophe Papazian

Université de Nice–Sophia Antipolis
Laboratoire I3S, UMR 6070 CNRS
2000 route des Lucioles, BP 121,
F-06903 Sophia Antipolis Cedex, France,
`Bruno.Martin@i3s.unice.fr`,
`Christophe.Papazian@i3s.unice.fr`

Abstract. We consider simulations of graph automata on surfaces by other graph automata obtained by local neighborhood transformations. We explain how to make such transformations, and what are the consequences on the topology of the simulated graph, the speed of the simulation and the memory size of simulating automata. In particular, we explain how to simulate hexagonal torus by square cylinder, and how to find the cheapest way (in terms of memory) to accelerate automata on trees.

Introduction

In this paper, we consider simulations between networks of automata arranged on graphs which are embedded on surfaces. The way to draw graphs on surfaces comes from combinatorial topology, an older name for algebraic topology which was addressed by Kuratowski [3].

Combinatorial topology (see [2]) was developed at first as a branch of geometry. The work of Euler and a number of nineteenth-century geometers on polyhedra is part of the development. Under the scope of this theory is also the study of surfaces. Surfaces are topological spaces in which every point has a neighborhood that is topologically equivalent to an open disk. The simplest example of a surface is the plane. Other objects can be constructed combinatorially by gluing disks together. With this kind of operation one can get a cylinder, a surface with boundary, or the torus which is the surface that results when both pairs of opposite sides of a rectangle are identified.

This kind of computational model has to be compared with cellular automata on Cayley graphs. This model shares the same underlying networks but is described in a completely different fashion. Instead of drawing the graph of the network on a surface, it is described by the means of a Cayley graph of a finitely presented group. The approach is more algebraic, which brings more constraints. Some authors considered simulations between cellular on Cayley graphs: Róka [9–12] proposed different simulations extended by Martin [4–6]. While Róka’s proofs are combinatorial or algebraic, Martin’s are algorithmic.

Some of the results can be imported from the Cayley graphs approach to the combinatorial topology approach and we will discuss their similarities and differences.

The paper is organized as follows. We introduce our model of computation in Section 1. Section 2 recalls related results from the Cayley graph approach. Section 3 introduces a few local transformations on the neighborhood and explains our results on infinite tree automata. Section 4 uses the local transformations to simulate finite graph automata.

1 Notations and definitions

1.1 Topological surfaces

We start with two surfaces with boundary before considering more classical topological surfaces. We consider the *cylinder* (see Fig. 1 left) which can be described

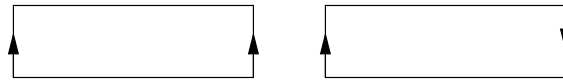


Fig. 1. Two surfaces with boundary; a cylinder (left) and a Möbius strip (right).

as a square in which top and bottom edges are given parallel orientations and the left and right edges are joined to place the arrow heads and tails into coincidence.

The Möbius strip (see Fig. 1 right) is a one-sided nonorientable surface obtained by cutting a closed band into a single strip, giving one of the two ends thus produced a half twist, and then reattaching the two ends .

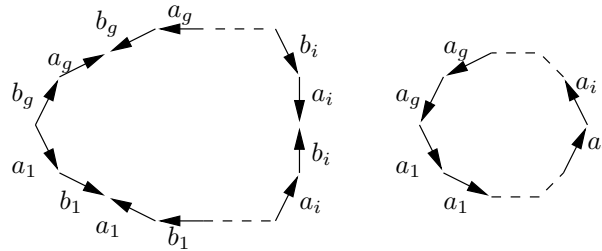


Fig. 2. Drawing of orientable surface (left) and nonorientable surface (right).

We then consider “classical” surfaces for drawing a picture of a graph in a way which will be detailed in section 1.2. There are two types of surfaces; *orientable* and *nonorientable*. The orientable surface of genus 0 and 1 are called respectively a *plane* and a *torus*. When we increase the genus of the orientable surface for $g \geq 2$, we obtain a g -handled torus, denoted by \vec{S}_g . The nonorientable surfaces of genus 1 and 2 are respectively a *projective plane* and a *Klein bottle*. Fig. 2 left describes an orientable surface of genus g in which for each pair of arcs sharing

the same label are joined together. Fig. 2 right describes a nonorientable surface of genus g (denoted by S_g in the sequel).

1.2 Embedding graphs on surfaces

A graph G is an ordered pair $G = (V, E)$ where V is a set of *vertices* (or *nodes*) and E is a set of *edges* which are pairs of distinct vertices.

A *path* in G is a sequence of vertices, each adjacent to the next. A *cycle* is a path containing at least three vertices such that the last vertex is adjacent to the first. Given x and y two vertices of a graph G , the *distance* between x and y is the length of a minimal path from x to y and is denoted by $d_G(x, y)$.

Since our goal is to embed graphs on surfaces and to associate a finite state machine to each vertex of the graph, we need some further definitions on graphs.

A graph is *planar* if it can be drawn in the plane so that no edges intersect or, equivalently, if it can be *embedded* in the plane. A nonplanar graph cannot be drawn without edge intersections. More generally, we consider in this paper graphs which are *embeddable* on an orientable surface \vec{S}_g , that is which can be drawn on \vec{S}_g without crossing edges.

The *faces* of a graph are the regions bounded by edges, including the outer, infinitely large regions (when existing). The *dual* of a given planar graph G has a vertex for each face of the graph and an edge for each edge joining two neighboring regions. Figure 3 illustrates the embedding of a hexagonal grid on a torus and the embedding of its dual graph on a torus. Vertices with the same number have to be identified as well as edges joining identical vertices.

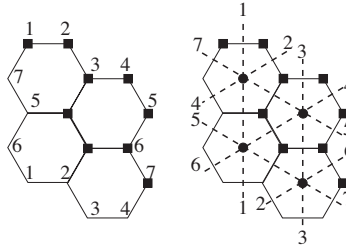


Fig. 3. Dual embeddings of a hexagonal graph on a torus. On the left, vertices with the same number are identified like on the right, the dotted edges with the same numbers.

1.3 Graph automata

A *partitioned graph automaton* (PGA for short) over a graph G is a 4-tuple $\mathcal{A} = (Q, G, N, \delta)$ for which we associate a finite state machine called a *cell* to each vertex of the graph G . The set Q denotes the finite set of the states, $G = (V, E)$ is a graph, N the neighborhood (including the cell itself and nodes at distance 1 together with a local numbering as described by Fig. 4) and $\delta : Q^{\#N} \rightarrow Q^{\#N}$ is the local transition function which updates the state of cell i at time t according to the states of (copies of) its neighbors at time $t - 1$, analogously

with the *partitioned CA* (PCA for short) introduced in [7]. We also define a distinguished state denoted by q , the *quiescent state* which has the property to remain quiescent by $\delta: \delta(q, \dots, q) = q$. Note that we only consider the von Neumann neighborhood (of one cell) defined as the set of vertices at distance at most one from the cell (thus including the cell itself) which we depict on Fig 4. The k -neighborhood (of a cell) is the set of vertices at distance at most k from the cell. Hence, as we need to know the neighborhood of a cell to compute one transition step, we need to know the k -neighborhood to compute k transition steps. We define a *configuration* of the CA as an application c which attributes a state to each cell. The set of all the configurations of a CA is denoted by $\mathbb{C} = Q^{\#N\#V}$ on which the *global function* Δ of the CA is defined by applying globally the local transition function.

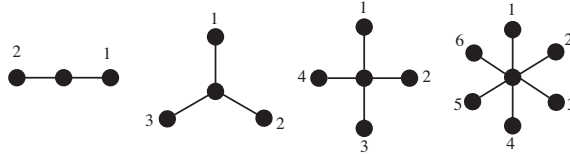


Fig. 4. Different kinds of neighborhoods; from left to right: N_2 , N_3 , N_4 and N_6 .

Definition 1 gives the formal statement of a simple PGA embedded on a cylinder (a ring of automata) and the behavior of the transition function is depicted on Fig. 5.

Definition 1. A 2 neighbors PGA on a cylinder is $\mathcal{A} = (Q, G, N_2, \delta)$ with set of states $Q = Q_2 \times Q \times Q_1$ with Q_2 a non-empty finite set of left internal states, Q a non-empty finite set of center internal states and Q_1 a non-empty finite set of right internal states, $G = C_n$ (the cycle graph with n vertices), N_2 the von Neumann neighborhood, and δ is the local transition function:

$$\delta : Q_1 \times Q \times Q_2 \rightarrow Q_2 \times Q \times Q_1$$

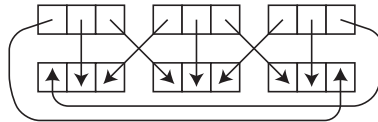


Fig. 5. 2 neighbors GA with 3 cells embedded on a cylinder ($\text{Cyl}N_2(3)$).

Definition 1 can be easily adapted to the neighborhoods depicted on Fig. 4.

In the sequel, we will consider some particular drawings of graphs on surfaces for which we introduce some notation. The first letter(s) denote the surface on which the graph will be embedded with the subscript denoting its genus (if relevant). The second letter gives the neighborhood of the graph according to Fig. 4. Last parameter(s) give the number of vertices of the graph. The simplest

one is the embedding of a cycle graph with n vertices on a cylinder as shown by Fig. 5. It will be denoted by $\text{Cyl}N_2(n)$. Next is the toroidal mesh which is the embedding of the Cartesian sum of two cycle graphs with respectively m and n vertices on a torus. It will be denoted by $\vec{S}_1N_4(m, n)$. We also consider the embedding of a hexagonal graph (resp. triangle, its dual graph) on a torus denoted by $\vec{S}_1N_6(m, n)$ (resp. $\vec{S}_1N_3(m, n)$). Observe that, for simplicity reason, the parameters of $\vec{S}_1N_3(m, n)$ are the same than $\vec{S}_1N_6(m, n)$. That is, we count the number of hexagons in each principal direction. We will also consider a generalisation as represented in Fig. 6: the embedding of a hexagonal graph (resp. triangle, its dual graph) on S_3 (the nonoriented surface of genus 3) denoted by $S_3N_6(m, n, k)$ (resp. $S_3N_3(m, n, k)$).

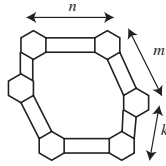


Fig. 6. S_3N_6 .

1.4 Simulation

Below, we propose the definition of a step by step simulation between two PGAs. It expresses that if a PGA A simulates each step of PGA B in τ units of time, there must exist a correspondence between the corresponding configurations:

Definition 2. Let \mathbb{C}_A and \mathbb{C}_B be the two sets of the configurations of PGAs A and B . We say that A simulates each step of B in time τ (and we note $B \xrightarrow{\tau} A$) if there exist a constant $\tau \in \mathbb{N}$ and two recursive functions $\kappa : \mathbb{C}_B \rightarrow \mathbb{C}_A$ and $\rho : \mathbb{C}_A \rightarrow \mathbb{C}_B$ such that $\kappa \circ \rho = \text{Id}$ and for all $c, c' \in \mathbb{C}_B$, then there exists $c'' \in \mathbb{C}_A$ such that if $c \vdash_B c'$, $\kappa(c) \vdash_A^{\tau} c''$ with $\rho(c'') = c'$, where \vdash_M denotes a global transition of PGA M and \vdash_M^t the t -th iterate of a global transition of PGA M .

Depending upon the value of τ , we say that the simulation is *elementary* if $\tau = 1$, *simple* if $\tau = O(1)$ and *general* for $\tau = O(f(\#c))$ with f denoting any given time-complexity function depending upon the size of the input c .

2 Related results

The results we recall in this section come from the Cayley graph approach. Except the definition of a local transition function (which is equivalent), they describe exactly the same objects but from a more algebraic point of view. We give here a statement of the results according to the definitions from section 1.

Theorem 1 (Róka [10]). $S_3N_6(r, p, q) \xrightarrow{1} \vec{S}_1N_4(m, n)$ with $m = |\alpha_1(p + r - 1) - \beta_1p|$, $n = |\alpha_2(p - 1) - \beta_2(p + q - 1)|$, $(p - 1)\alpha_1 = \text{lcm}(p - 1, p + q - 1)$,

$(p + r - 1)\alpha_2 = lcm(p + r - 1, p)$, $(p + q - 1)\beta_1 = lcm(p - 1, p + q - 1)$ and $p\beta_2 = lcm(p + r - 1, p)$.

Theorem 2 (Martin [4]). $\vec{S}_1 N_4(m, n) \stackrel{3 \cdot \min\{m, n\} + O(1)}{\prec} CylN_2(mn)$.

And, by combining theorem 1 and theorem 2, one can get:

Corollary 1. $S_3 N_6(r, p, q) \stackrel{3 \cdot \min\{m, n\} + O(1)}{\prec} CylN_2(mn)$ where m and n are those of Theorem 1.

Theorem 3. $\vec{S}_1 N_4(m, n) \stackrel{\Theta(n+m)}{\prec} CylN_2(mn)$ if and only if $n \equiv 2 \pmod{m}$; in this case the number of copies of each cell is minimal.

Theorem 3 improves the time-complexity of theorem 2. We have minimized the number of copies requested to simulate the behavior of a torus of $n \times m$ automata by a ring of $n \cdot m$ automata. Observe that this number of copies cannot be further improved. It is thus the minimal number of copies requested to complete this task. Theorem 3 forbids some values of n and m . However, for these forbidden values, one can use theorem 2.

3 Neighborhood transformations

We can see the transformations of neighborhoods as local transformations. Such transformations allow to handle the same computation with very different automata, without changing most of the topological properties of the network. We present here different local transformations and some results about their consequences on computations.

3.1 Homogeneous networks and splittings

Homogeneous networks are networks where all vertices have the same number of neighbors. No other hypothesis are made about the networks. The main transformation on such networks is splitting.

Definition 3. A splitting s_G is a local transformation that replaces each single node by a graph G with the same number of outgoing edges. The splitting is regular if the homogeneous property is conserved.

Fig. 7 shows two simple regular splits. The 2-split transforming a $2n$ -node into two $n + 1$ -nodes and the multisplit transforming a n -node into n 3-nodes.

If we consider simulations of computations, we obtain this first result.

Lemma 1. Any network \mathcal{N} can be simulated by any other network $s_G(\mathcal{N})$ obtained by application of a split. The factor of deceleration equals one plus the diameter of the subgraph G .

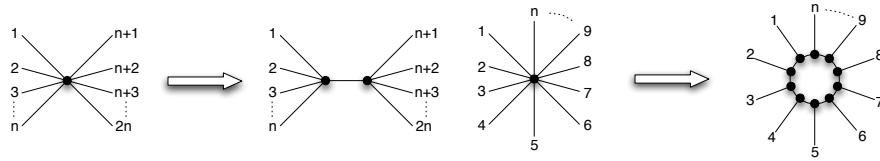


Fig. 7. 2-split and multisplit

Proof. The diameter of a graph is the maximum length of shortest paths between any two vertices of the graph. Obviously, each subgraph G needs one step of computation to “read” the states of neighbors subgraphs. Then, d steps are needed to obtain, in each nodes of subgraphs the complete information on the neighborhood to compute the simulated transition. Hence one can compute a simple simulation with a slowdown factor of $d + 1$.

Hence, we can simulate complex homogeneous networks with a high degree of connectivity with bigger but less connected networks.

The reverse operation is much more difficult, due to the fact that we need some special property of the network for merging. In fact, merging (the reverse operation of splitting) is only possible on graphs that can be obtained from another one by splitting. But how much is the acceleration factor ?

Lemma 2. *Any network \mathcal{N} can be simulated by any other network $m_G(\mathcal{N})$ obtained by application of a merge. The factor of acceleration equals one: in the general case, there is no speedup.*

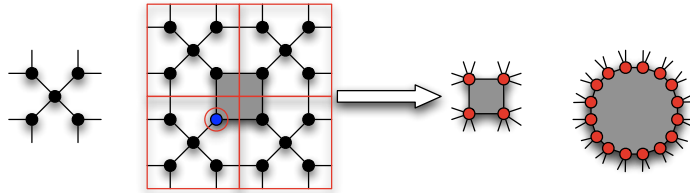


Fig. 8. Acceleration problem

Proof. As we consider only von Neumann neighborhoods, the k -neighborhood of a node v is the set of all vertices at distance at most k from v . To simulate k steps of computation, an automaton needs to know the states of all vertices of the k -neighborhood of the simulated vertex.

Fig. 8 shows how such pieces of information about k -neighborhood can be difficult to gather in arbitrary networks. The subgraph on the left is G , our merging pattern. When we apply the merge on the large network, we obtain a four vertex network. But one can remark that the 2-neighborhood of the blue vertex is embedded in the 2-neighborhood of the macro-vertex in the resulting network. Hence, we cannot really apply a simple speed-up, as we need two steps

to gather the 2-neighborhood of all vertices in each subgraph G . This is due to the size 4 of the gray face. On the right, there is a n -face, where the speed-up will be even more difficult. In arbitrary networks, arbitrary large faces occurs, that can prevent any possible acceleration.

However, many networks can be simulated with a good speedup factor by merging vertices. Hence, we must remember that we use some regular property of those networks and not only merging to obtain an acceleration.

Definition 4. *The internal path length of a subgraph with outgoing edges is the shortest path between two different outgoing edges. If there is a vertex with two outgoing edges, the internal path length is zero.*

Theorem 4. *Any network \mathcal{N} can be simulated by any other network $m_G(\mathcal{N})$ obtained by application of a merge. The factor of acceleration is at least equals to one plus the internal path length of the subgraph G .*

Proof. Let i be the internal path length of G . In $m_G(\mathcal{N})$, the neighborhood of any vertex v contains the $(1+i)$ -neighborhood of any vertices of \mathcal{N} simulated by v . This is due to fact that any path of length $1+i$ can not reach outgoing edges of neighborhood subgraph G .

3.2 Efficient merges

Theorem 4 only gives a lower bound. We show now several ways to use regularity to find efficient merge for accelerated simulation.

n-ary tree In a infinite regular n -ary tree T_n , one can merge using any finite tree t . We consider the oriented tree, each vertex having one father, and n sons.

The acceleration factor will only depend upon the smallest path from the root of t to a descendants not in t . Hence we only consider complete n -ary tree of height h as t , that we note t_n^h .

$m_{t_n^h}(T_n) = T_{n^{h+1}}$, and the k -neighborhood in the merged tree contains the simulated $(1+(k-1)h)$ -neighborhood. Hence, for any $\epsilon > 0$, we can simulate T_n by $T_{n^{h+1}}$, with an acceleration factor of $h - \epsilon$. Each automaton reads its k -neighborhood (with $k \geq \frac{h-1}{\epsilon}$) using k time steps, and simulate $1+(k-1)h$ time steps of T_n . The factor of acceleration is $\frac{1+(k-1)h}{k} = h - \frac{h-1}{k} \geq h - \epsilon$.

Memory usage Hence, when one wants to simulate T_n at speed s , one can choose any $h > s$ to merge using t_n^h , and then $k = \lceil \frac{h-1}{h-s} \rceil$ will be the size of the neighborhood in the new network that we read before simulating several steps of computation. But how big is the simulating automaton ? t_n^h contains $\frac{n^{h+1}-1}{n-1}$ vertices. And the k -neighborhood of a vertex in T_n contains $1+(n+1)\frac{n^k-1}{n-1}$

vertices (only 1 vertex if $k = 0$). Let s the speed we want to obtain, and h the height of t_n^h that we used for merging, the new automaton must contain at least:

$$m = \frac{n^{h+1} - 1}{n - 1} \left(1 + (n^{h+1} + 1) \frac{n^{(h+1)(\lceil \frac{h-1}{h-s} \rceil - 1)} - 1}{n^{h+1} - 1} \right)$$

copies of simulated automata. It means that a simulating automaton must remember the states of m different simulated automata to compute the simulation. By minimizing this formula, we obtain the best height to achieve the simulation at given speed with a minimum memory size for the new automaton. The minimal height h doesn't depend of n , and the height h that minimizes memory is obtained for k -neighborhood $k = 2$ then $h = 2s - 1$. This is the best way to simulate merged infinite regular trees. In this case, memory usage grows as n^{4v} , which is quite fast. Hence merging is better than waiting !

Grids Merging grids is much more complex, as one can use any tiling patterns to merge a grid, and this is an open question to know if a pattern tiles a grid [1]. Some tilings are surprisingly much more efficient than other ones. We give an example in figure 9. On the left, we merge using squares, and as we see for trees, we can achieve any speed $2 - \epsilon$ (epsilon is expensive to minimize as in trees), on the right we merge using bottom pattern, and we can achieve a speed of 2 with $k = 2$ which is optimal for a pattern of 4 vertices !

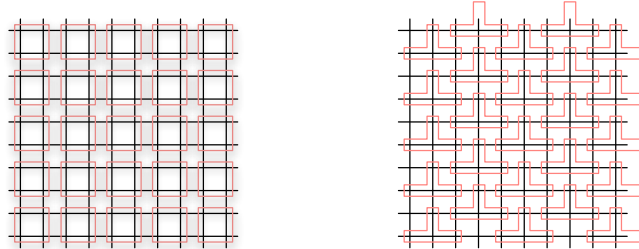


Fig. 9. Simple and optimal merge of the grid for 4-vertices patterns.

The results on trees hold on grids: merging is better than waiting. For example, when merging grid using squares, it is better to use big squares and read the 2-neighborhood than using smaller squares and read larger neighborhood. The optimal memory usage (for a simulation at speed s) growths as $4s$.

4 Finite simulations

In this section, we consider finite simulation i.e. of finite PGA embedded on surfaces. We start with three simulation lemmas which allow us to change the neighborhood of the PGA. Lemma 3 explains how to simulate the behavior of any PGA with 6 neighbors by a PGA with only 4. The main idea is to cut a

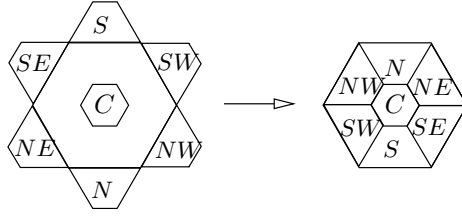


Fig. 10. Transition of a N_6 -PGA.

hexagon into two pieces and by adding a new part state (R for the left part and L for the right part). The simulation depicted on Fig. 11 is as follows:

1. gather the missing neighborhood information and pack it in the added part state (Fig. 11 left);
2. simulate one step of h according to the new neighborhood (Fig. 11 right).

Lemma 3. N_6 -PGA $\stackrel{2}{\sim}$ N_4 -PGA

Proof. According to [8], a N_6 -PGA is defined as (Q, G_6, N_6, h) with $Q = (C, N, NE, SE, S, SW, NW)$ sets of center, north, north-east, south-east, south, south-west and north-west part states. G_6 is the graph of hexagons which is embedded on a surface, N_6 is as depicted on Fig 4. The local function h is a mapping (see Fig. 10):

$$h : C \times S \times SW \times NW \times N \times NE \times SE \rightarrow C \times N \times NE \times SE \times S \times SW \times NW$$

A N_4 -PGA is defined as (Q, G_4, N_4, σ) with $Q = (C, U, R, L, D)$ sets of center, up, right, left and down part states. G_4 is a toroidal mesh, N_4 as depicted on Fig 4. The local function σ is a mapping (see Fig. 11):

$$\sigma : C \times D \times L \times U \times R \rightarrow C \times U \times R \times D \times L$$

To simulate the behavior of a N_6 -PGA by a N_4 -PGA, we distinguish periodically two cells: one which gathers the contents of the right part of the hexagon and, respectively, one which gathers the left part of the hexagon. The first rule of σ is:

- (1) $(C, D, L, U, R) \mapsto (C, U, R = (D, U, R), D, L)$ gathering left part
- (2) $(C, D, L, U, R) \mapsto (C, U, R, D, L = (D, L, U))$ gathering right part

After these rules, the contents of the R part is for (1) (D, U, R) which contains a copy of (SE, N, NE) and for the L part, (S, SW, NW) . After that, the center part C has all the necessary information to simulate one transition step of h .

Since the transformation is local, we get N_6 -PGA $\stackrel{2}{\sim}$ N_4 -PGA

Observe that this simulation corresponds to a splitting introduced in Definition 3.

Lemma 4 claims that a 6 neighbors PGA can be simulated by a 3 neighbors PGA (and conversely). It was proved by Róka by using Cayley graphs. But, since it is a local simulation, it can be used in our context as well.

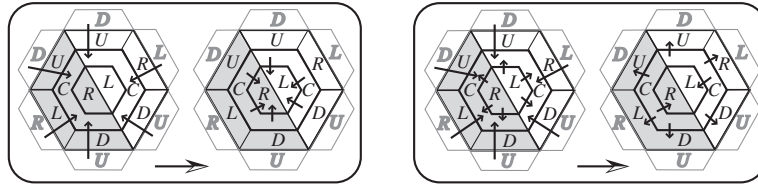


Fig. 11. Simulating N_6 -PGA by N_4 -PGA.

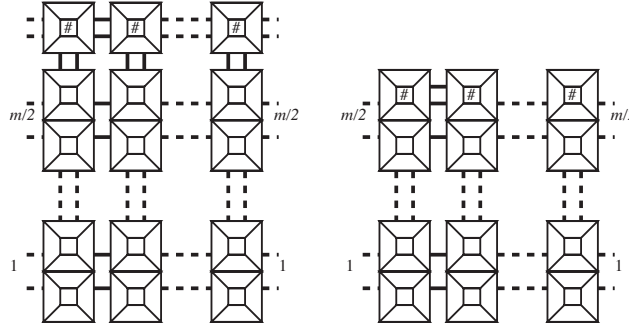


Fig. 12. Simulating a PGA on a torus by a PGA on a cylinder (even and odd m).

Lemma 4 (Róka [12]). N_6 -PGA $\stackrel{1}{\sim}$ N_3 -PGA (and conversely).

Below, we propose a series of finite simulation results. The first one is lemma 5 which proposes a simulation of a N_4 -PGA embedded on a torus by a N_4 -PGA embedded on a Cylinder. In order to do this, one needs to cut the torus along one of its Jordan curves.

Lemma 5. $\vec{S}_1 N_4(n, m) \stackrel{1}{\sim} Cyl N_4(n, \lceil \frac{m}{2} \rceil)$.

Proof (sketch). We consider a N_4 -PGA with (m, n) nodes embedded on the torus with m being the height and n the width. We “cut” all the connections along the width and we fold the resulting cells on the middle. We add a “dummy” cell (with symbol \sharp) on the top depending upon the parity of m as pictured by Fig. 12. It is not difficult to design the local transition function of the new PGA.

Theorem 5 proposes two simulations of a N_6 -PGA embedded on a torus. The first one by a N_4 -PGA embedded on a torus and the other by a N_4 -PGA embedded on a cylinder. The results are easily obtained from the previous lemmas.

Theorem 5.

$$\begin{aligned} \vec{S}_1 N_6(n, m) &\stackrel{2}{\sim} \vec{S}_1 N_4(2n, m) \\ \vec{S}_1 N_6(n, m) &\stackrel{2}{\sim} Cyl N_4(n, m) \end{aligned}$$

These results can be combined with the results from the Cayley graph approach and give plenty of other results as, for instance, it is straightforward to replace N_6 by N_3 in theorem 5.

5 Conclusion

We have proposed a new approach for describing networks of automata which were commonly addressed by cellular automata on Cayley graphs. The combinatorial topology approach permits to design simulations with interesting time complexities. Results from the Cayley graph approach describing local transformations (which preserve the property of being a Cayley graph) can be imported giving new simulation results.

Moreover, the links between tilings, merges and simulations show the deepness of this subject, and we hope to make new links between acceleration and undecidability problems in tilings.

Whereas turing machines world have strong and efficient simulation and acceleration theorems, the world of graph automata has a lack of equivalent results due to the subtil interaction between computation and graphs. This is a major scientific goal for our more and more interconnected world.

References

1. D. Beauquier and M. Nivat. Tiling the plane with one tile. In *Symposium on Computational Geometry*, pages 128–138, 1990.
2. M. Henle. *A combinatorial introduction to topology*. Dover Publication, 1979.
3. K. Kuratowski. *Introduction à la théorie des ensembles et à la topologie*. Institut de Mathématiques de l'Université de Genève, 1966.
4. B. Martin. Embedding torus automata into a ring of automata. *Int. Journal of Found. of Comput. Sc.*, 8(4):425–431, 1997.
5. B. Martin. A simulation of cellular automata on hexagons by cellular automata on rings. *Theoretical Computer Science*, 265:231–234, 2001.
6. B. Martin and C. Peyrat. A single copy minimal-time simulation of a torus of automata by a ring of automata. submitted.
7. K. Morita and M. Harao. Computation universality of one-dimensional reversible (injective) cellular automata. *Trans. IEICE Japan*, E(72):758–762, 1989.
8. K. Morita, M. Margenstern, and K. Imai. Universality of reversible hexagonal cellular automata. In *MFCS'98 Satellite Workshop on Frontiers between Decidability and Undecidability*, Brno, 1998.
9. Zs. Róka. One-way cellular automata on Cayley graphs. In *Proc. FCT'93*, number 710 in Lecture Notes in Computer Science, pages 406–417. Springer Verlag, 1993.
10. Zs. Róka. *Automates cellulaires sur graphes de Cayley*. PhD thesis, École Normale Supérieure de Lyon, 1994.
11. Zs. Róka. One-way cellular automata on Cayley graphs. *Theoretical Computer Science*, 132(1–2):259–290, 1994.
12. Zs. Róka. Simulations between cellular automata on cayley graphs. *Theoretical Computer Science*, 225:81–111, 1999.