

---

## NEIGHBORHOOD TRANSFORMATIONS ON GRAPH AUTOMATA

BRUNO MARTIN AND CHRISTOPHE PAPAZIAN

Université de Nice–Sophia Antipolis, I3S, UMR 6070 CNRS, 2000 route des Lucioles, BP 121,  
F-06903 Sophia Antipolis Cedex.

*E-mail address:* {Bruno.Martin|Christophe.Papazian}@unice.fr

---

**ABSTRACT.** We consider simulations of graph automata. We introduce two local transformations on the neighborhood: splitting and merging. We explain how to use such transformations, and their consequences on the topology of the simulated graph, the speed of the simulation and the memory size of simulating automata in some cases. As an example, we apply these transformations to graph automata embedded on surfaces and we link our results with some simulation results between cellular automata on Cayley graphs.

### 1. 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 [4].

Combinatorial topology (see [3]) 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 in a combinatorial way by gluing disks together. With this kind of operation one gets 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 networks of automata has to be compared with cellular automata on Cayley graphs. Both models share the same underlying networks but are described in a completely different fashion. Instead of drawing the graph of the network on a surface, it is defined by the Cayley graph of a finitely presented group. The approach, more algebraic, brings more constraints. Some authors already considered simulations between cellular automata on Cayley graphs: Róka [10, 11, 12, 13] proposed different simulations extended by Martin [5, 6, 7].

---

*Key words and phrases:* Graph automata, Cellular automata on Cayley graphs, algorithmics, topology.

(This work has been supported by the Interlink/MIUR project “Cellular Automata: Topological Properties, Chaos and Associated Formal Languages” and by the french ANR programme Sycomore.)

Some 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 2. Section 3 presents our local transformations on the neighborhood and gives some examples. Section 4 uses the local transformations to simulate finite graph automata embedded on surfaces.

## 2. Notation and definitions

We start with two surfaces with boundary: the *cylinder* and the *Moebius strip* (see Fig. 1). The *cylinder* which can be described as a square in which top and bottom edges



Figure 1: Two surfaces with boundary; a cylinder (left) and a Moebius strip (right).

are given parallel orientations and the left and right edges are joined to place the arrow heads and tails into coincidence. The Möbius strip is a one-sided non-orientable 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.

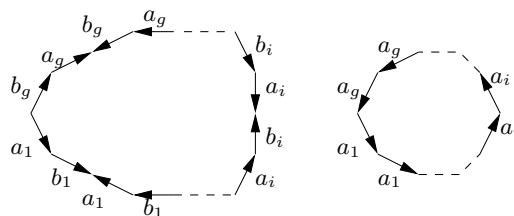


Figure 2: Drawing of orientable surface (left) and non-orientable surface (right).

We then consider “classical” surfaces for drawing a picture of a graph (cf. section 2.1). There are two types of surfaces: *orientable* and *non-orientable*. The orientable surface of genus 0 and 1 are respectively called a *plane* and a *torus*. When we increase the genus  $g$  of the orientable surface for  $g \geq 2$ , we obtain  $\vec{S}_g$  a  $g$ -handled torus. The non-orientable 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 each pair of edges sharing the same label are joined together. Fig. 2 right describes  $S_g$ , a non-orientable surface of genus  $g$ .

In the rest of the paper we will keep the notations of combinatorial topology for the surfaces we consider. Thus, the usual ring becomes a cylinder (abbreviated by *Cyl*), a “usual” torus remains a torus (abbreviated by  $\vec{S}_1$ ).

### 2.1. 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* is a sequence of vertices, each adjacent to the next. A *cycle* is a path with at least 3 vertices such that the last vertex is adjacent to the first. Given  $x$  and  $y$  two vertices of  $G$ , the *distance* between them is the length of a minimal path from  $x$  to  $y$ .

Since our goal is to embed regular graphs (i.e. isomorphic to Cayley 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.

When a graph is drawn without any crossing, any cycle that surrounds a region without any edge reaching from the cycle inside to such region forms a *face*, including the outer, infinitely large regions (when existing). Observe that the notion of face is independent from the embedding [2].

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. Fig. 3 illustrates the embedding of an hexagonal grid on a torus and the embedding of its dual on a torus. Vertices with the same number have to be identified as well as edges joining identical vertices.

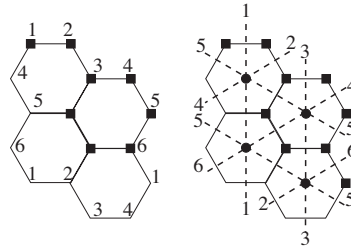


Figure 3: Dual embeddings of an hexagonal graph on a torus. On the left, vertices with the same number are identified like on the right, the dotted edges with the same numbers.

### 2.2. 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; the numbering gives an ordering of the neighbors that will be used by the local transition function) 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 [8]. In a PGA (as well as in a PCA), the states are partitioned according to the neighborhood and only the relevant pieces of states are available to any state. Sub-states are gathered

to form only one state to be updated. That is, each state of a PGA is a  $\#N$ -tuple, each tuple contains information for a specific neighbor. This model simplifies the simulations we are considering in section 2.3. We define a distinguished state  $q$ , the *quiescent state* that verifies  $\delta(q, \dots, q) = q^{\#N}$ . Note that we only consider the radius 1 neighborhood (of one cell) defined as the set of vertices at distance at most one from the cell (thus including the cell itself) that 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 PGA as an application  $c$  which attributes a state to each cell. The set of all the configurations of a PGA is denoted by  $\mathbb{C} = Q^{\#N\#V}$  on which the *global function*  $\Delta$  of the PGA is defined by applying globally the local transition function.

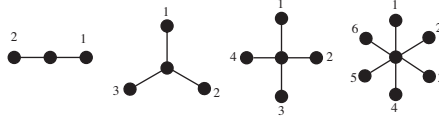


Figure 4: Different kinds of neighborhoods; from left to right:  $N_2$ ,  $N_3$ ,  $N_4$  and  $N_6$ .

Definition 2.1 gives the formal statement of a simple PGA embedded on a cylinder and the behavior of the transition function is depicted on Fig. 5.

**Definition 2.1.** A 2-neighbor PGA on a cylinder is  $\mathcal{A} = (Q, G, N_2, \delta)$  with set of states  $Q = L \times C \times R$ .  $L, C$  and  $R$  are all non-empty subsets of the same set of states  $Q'$  with  $L$  the set of left internal states,  $C$  the set of center internal states and  $R$  the set of right internal states,  $G = C_n$  (the cycle graph with  $n$  vertices),  $N_2$  the von Neumann neighborhood, and  $\delta$  the local transition function:

$$\delta : R \times C \times L \rightarrow L \times C \times R$$

A configuration of  $\mathcal{A}$  is a mapping  $\mathbb{Z}_n \rightarrow L \times C \times R$ . The set of all configurations is denoted by  $\mathbb{C}$ . We denote by LEFT (CENTER, RIGHT resp.) the projection function which picks out the left (resp. center, right) element of a triple in  $L \times C \times R$ . The global function is  $\Delta(c)(i) = \delta(\text{RIGHT}(c(i-1)), \text{CENTER}(c(i)), \text{LEFT}(c(i+1)))$  where  $i, i-1$  and  $i+1$  are integers modulo  $n$ .

Definition 2.1 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) denotes 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 gives the number of vertices of the graph. The simplest one is the embedding of a cycle graph with  $n$  vertices on a cylinder (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<sup>1</sup> 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 an 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  $\vec{S}_1N_6(m, n)$ . Indeed, we count the number of hexagons in each principal

<sup>1</sup>The cartesian sum is often called cartesian product but the definitions differ [2].

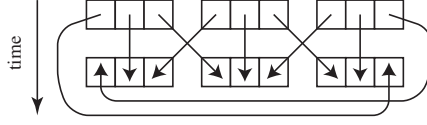


Figure 5: Two configurations of a 2-neighbor 3 cells PGA embedded on a cylinder  $\text{Cyl}N_2(3)$ .

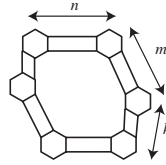


Figure 6:  $S_3N_6$ : embedding of an hexagonal graph in a 3-handled torus.

direction. We will also consider a generalization as represented in Fig. 6: the embedding of an hexagonal graph (resp. triangle, its dual graph) on  $S_3$  (the non-oriented surface of genus 3) denoted by  $S_3N_6(m, n, k)$  (resp.  $S_3N_3(m, n, k)$ ). The regularity and extendability of  $S_3N_6(m, n, k)$  comes from its group representation and was considered by Róka in her work on cellular automata on Cayley graphs.

Since we are dealing with graphs which are (cellularly) embedded into surfaces, there is no need here to fully define the interconnection pattern. All the graphs we consider being regular, the way they are connected is defined by the neighborhood (cf. Fig. 4).

### 2.3. 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  $\tau$  units of time, there must exist effective applications between the corresponding configurations:

**Definition 2.2.** Let  $\mathbb{C}_A$  and  $\mathbb{C}_B$  be the two sets of PGA configurations  $A$  and  $B$ . We say that  $A$  simulates each step of  $B$  in time  $\tau$  (and we note  $B \stackrel{\tau}{\sim} A$ ) if there exists 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  $c, c' \in \mathbb{C}_B$ , there exists  $c'' \in \mathbb{C}_A$  such that if  $c' = \Delta_B(c)$ ,  $c'' = \Delta_A^\tau(\kappa(c))$  with  $\rho(c'') = c'$ , where  $\Delta_M$  denotes a global transition of PGA  $M$  and  $\Delta_M^t$  the  $t$ -th iterate of a global transition of PGA  $M$ .

Depending upon the value of  $\tau$ , 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 on  $c$ , the size of the input.

## 3. Neighborhood transformations

Neighborhood transformations can be seen as local transformations. Such transformations allow to handle the same computation with slightly different automata, without changing the major topological properties of the GA. We present two local transformations: *splitting* and *merging* and we give some consequences on the computations.

### 3.1. Homogeneous GAs and splittings

Homogeneous GAs are networks where all the vertices have the same number of neighbors. No other assumption is made. We first introduce the *splitting*.

**Definition 3.1.** A splitting  $s_G$  is a local transformation that replaces simultaneously each single vertex by a subgraph  $G$  with the same number of outgoing edges (edges that do not belong to the subgraph but that link it to the network). The splitting is regular if the GA remains homogeneous (if  $G$  is homogeneous).

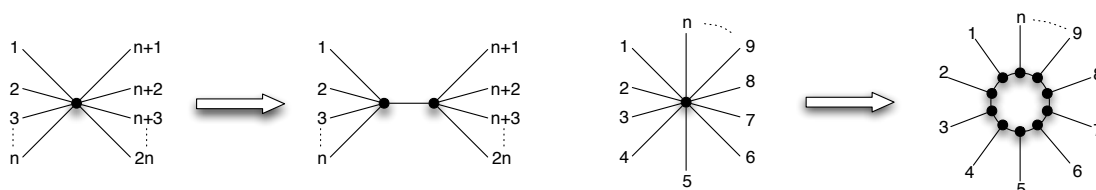


Figure 7: 2-split and multisplit.

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, we obtain Lemma 3.2.

**Lemma 3.2.** Any GA  $\mathcal{N}$  can be simulated by any other GA  $s_G(\mathcal{N})$  obtained by application of a split. The bound on the factor of deceleration equals one plus the diameter of the subgraph  $G$ ; with our notation,  $\mathcal{N} \stackrel{d+1}{\prec} s_G(\mathcal{N})$ .

*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 neighbor subgraphs. Then,  $d$  steps are required to obtain, in each nodes of the 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 GAs with a high degree of connectivity with bigger but less connected GAs.

### 3.2. Example of a 2-split

Lemma 3.3 explains the simulation of any 6-neighbor PGA by a 4-neighbor PGA.

**Lemma 3.3.**  $N_6\text{-PGA} \stackrel{2}{\prec} N_4\text{-PGA}$ .

The idea is to cut the hexagon using a 2-split for transforming a 6 node into two 4 nodes and by adding a new part state called a *layer* ( $R$  for the left part and  $L$  for the right part). The simulation depicted on Fig. 9 is as follows:

- (1) gather the missing neighbors information; pack it in the layer (Fig. 9 left);
- (2) simulate one step of  $h$  according to the new neighborhood (Fig. 9 right).

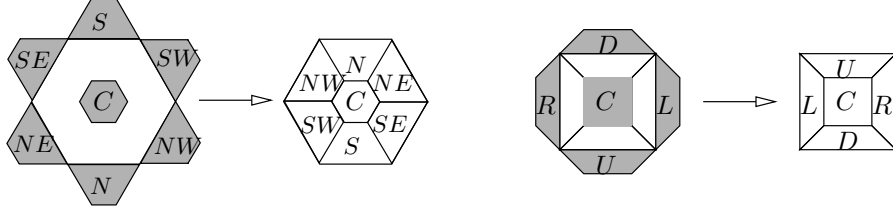


Figure 8: Transition of a  $N_6$ -PGA (left) and of a  $N_4$ -PGA (right).

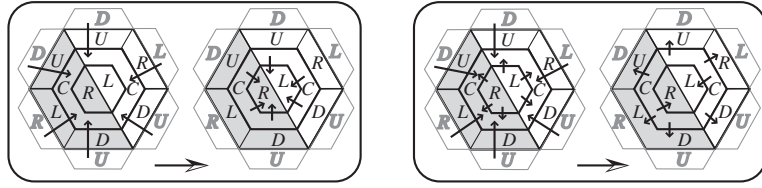


Figure 9: Simulating  $N_6$ -PGA by  $N_4$ -PGA: gather neighbor informations (left) and simulate one transition step (right).

*Proof.* A  $N_6$ -PGA [9] is  $(Q, G_6, N_6, h)$  with  $Q = C \times N \times NE \times SE \times S \times SW \times NW$  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. 8):

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

A  $N_4$ -PGA is  $(Q, G_4, N_4, \sigma)$  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 on Fig 4. The local function  $\sigma$  is a mapping:  $\sigma : C \times D \times L \times U \times R \rightarrow C \times U \times R \times D \times L$  (Fig. 8 right).

To simulate 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  $\sigma$  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 part  $C$  has all the necessary information to simulate one transition step of  $h$  (Fig. 9). The factor of deceleration is 2 as stated in Lemma 3.2.  $\blacksquare$

Below, we also recall Lemma 3.4 which states 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 transformation, it will be used later.

**Lemma 3.4** (Róka [13]).  $N_6$ -PGA  $\stackrel{1}{\sim}$   $N_3$ -PGA (and conversely).  $\blacksquare$

### 3.3. Homogeneous GAs and merging

Merging, the converse operation of splitting is more difficult. It is due to the fact that we need some special property of the GA for merging. Actually, merging is only possible if one of the graphs can be obtained by splitting from the other one (finding if a graph can be obtained by splitting seems to be a complex NP problem). But what is the acceleration factor?

**Lemma 3.5.** *Any GA  $\mathcal{N}$  can be simulated by any other GA  $m_G(\mathcal{N})$  obtained by application of a merge. The factor of acceleration equals one: in the worst case, there is no speedup.*

*Proof.* As we consider only radius 1 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. 10 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 GA, we obtain a four vertices GA. But one can remark that the 2-neighborhood of the circled vertex is not contained in the 1-neighborhood of the macro-vertex in the resulting GA. 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 grey face. On the right, there is a  $n$ -face, where the speed-up will be even more difficult. In arbitrary networks, arbitrary large faces occur. Hence there are arbitrary long paths that are not shrunk by the merge, preventing any acceleration (due to the “speed of light” limit inherently present in any automata network). Obviously, in some finite case, we can use more complex acceleration techniques, but the worst case can possibly occur. ■

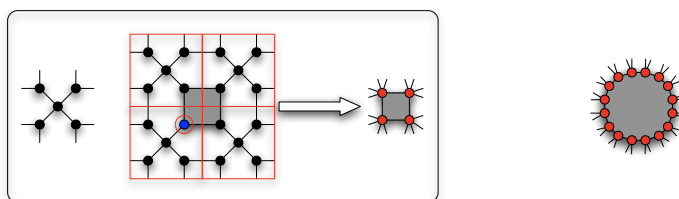


Figure 10: Acceleration problem.

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. The theorem 3.7 is a refinement of the lemma 3.5. We obtain a more precise bound using the internal path length.

**Definition 3.6.** 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 3.7.** *Any GA  $\mathcal{N}$  can be simulated by any other GA  $m_G(\mathcal{N})$  obtained by application of a merge. The factor of acceleration is at least 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 vertex of  $\mathcal{N}$  simulated by  $v$ . This is due to the fact that any path of length  $1+i$  cannot reach outgoing edges of neighborhood subgraph  $G$ . ■

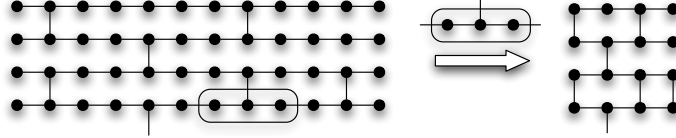


Figure 11: An efficient merging.

The figure 11 shows a merge with a subgraph  $G$  with an internal path length of 1.

### 3.4. Some efficient merges

Theorem 3.7 only gives a lower bound. We show now several ways to use regularity to find efficient merges to speed-up the simulations.

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

The acceleration factor only depends upon the smallest path from the root of  $t$  to a descendant 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  $\varepsilon > 0$ , we can simulate  $T_n$  by  $T_{n^{h+1}}$ , with an acceleration factor of  $h - \varepsilon$ . Each automaton reads its  $k$ -neighborhood (with  $k \geq \frac{h-1}{\varepsilon}$ ) using  $k$  time steps, and simulates  $1+(k-1)h$  time steps of  $T_n$ . The factor of acceleration is  $\frac{1+(k-1)h}{k} = h - \frac{h-1}{k} \geq h - \varepsilon$ .

3.4.2. *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 which is the size of 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  $s$  be 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  $h$  does not depend on  $n$ , and the height  $h$  that minimizes the memory size 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

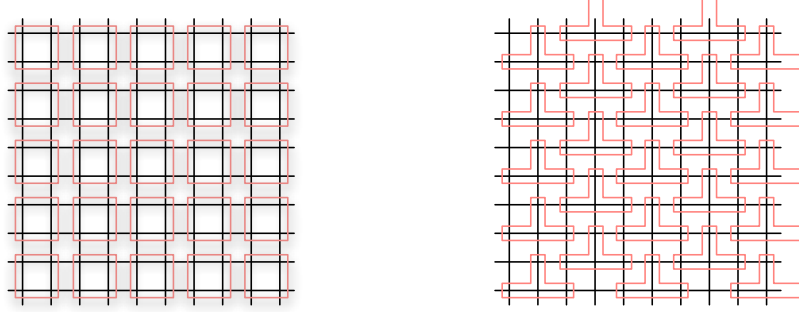


Figure 12: Simple (left) and optimal (right) merges of the grid for 4-vertices patterns.

usage grows as  $n^{4s}$ , which is quite fast. Hence merging is better than waiting (intuitively, merging allows several vertices to be merged and they can share memory, whereas waiting needs to copy on each vertex redundant pieces of information).

**3.4.3. Grids and toroidal meshes.** Merging grids is more complex, as any tiling pattern can be used to merge a grid, and it is an open question to know if a pattern tiles a grid [1]. Some tilings are surprisingly more efficient than others. Consider Fig. 12; on the left, we merge with squares, and as for trees, we can achieve any speed  $2 - \varepsilon$  ( $\varepsilon$  is expensive to minimize as in trees). On the right we merge using a pattern whose shape resembles a bottom symbol, and we achieve a speed of 2 with  $k = 2$  which is optimal for a pattern of 4 vertices.

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

## 4. Application to the simulations of GA on surfaces

We apply the neighborhood transformations introduced in section 3 to PGA embedded on surfaces. The simulations we construct have tight relations with some results on cellular automata on Cayley graphs recalled in the next section.

### 4.1. Related results

The results we present here come from the Cayley graph approach. Except for 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 for finite PGAs embedded on surfaces as defined in Section 2.1.

**Theorem 4.1** (Róka [11]).  $S_3N_6(r, p, q) \stackrel{1}{\prec} \vec{S}_1N_4(m, n)$  with  $m = |\alpha_1(p + r - 1) - \beta_1p|$ ,  $n = |\alpha_2(p - 1) - \beta_2(p + q - 1)|$ ,  $(p - 1)\alpha_1 = \text{lcm}(p - 1, p + q - 1)$ ,  $(p + r - 1)\alpha_2 = \text{lcm}(p + r - 1, p)$ ,  $(p + q - 1)\beta_1 = \text{lcm}(p - 1, p + q - 1)$  and  $p\beta_2 = \text{lcm}(p + r - 1, p)$ . ■

**Theorem 4.2** (Martin [5]).  $\vec{S}_1N_4(m, n) \stackrel{3. \min\{m, n\} + O(1)}{\prec} \text{Cyl}N_2(mn)$ . ■

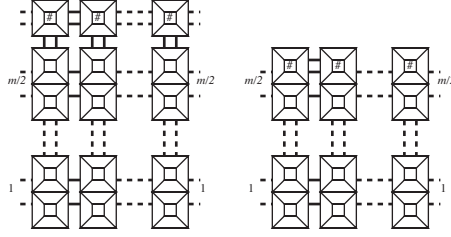


Figure 13: Simulating a PGA on a torus by a PGA on a cylinder (even and odd  $m$ ).

And, by combining Theorem 4.1 and Theorem 4.2, we get:

**Corollary 4.3.**  $S_3N_6(r, p, q) \stackrel{3 \cdot \min\{m, n\} + O(1)}{\prec} CylN_2(mn)$  where  $m$  and  $n$  are those of Theorem 4.1. ■

**Theorem 4.4** (Martin-Peyrat [7]).  $\vec{S}_1N_4(m, n) \stackrel{\Theta(n+m)}{\prec} CylN_2(mn)$  if and only if  $n \equiv 2 \pmod{m}$ ; in this case the number of copies of each cell is minimal. ■

Theorem 4.4 improves the time-complexity of Theorem 4.2. We have minimized the number of copies requested to simulate the behavior of a torus of  $n \times m$  automata by a cylinder 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 4.4 forbids some values of  $n$  and  $m$ . However, for those prohibited values, one can use Theorem 4.2.

## 4.2. New simulation results

Below, we propose a series of finite simulation results. The first (Lemma 4.5) proposes a simulation of a  $N_4$ -PGA embedded on a torus by a  $N_4$ -PGA embedded on a cylinder. To do this, we need to cut the torus along one of its Jordan curves.

**Lemma 4.5.**  $\vec{S}_1N_4(n, m) \stackrel{1}{\prec} CylN_4(n, \lceil \frac{m}{2} \rceil)$ .

(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. This construction is analogous with the simulation of a Turing machine with a bi-infinite tape by a Turing machine with an (simply) infinite tape. We add a “dummy” cell (with symbol  $\sharp$ ) on the top depending upon the parity of  $m$  (see Fig. 13). It is not difficult to design the local transition function of the new PGA. ■

Theorem 4.6 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 4.6.**  $\vec{S}_1N_6(n, m) \stackrel{2}{\prec} \vec{S}_1N_4(2n, m)$ , and  $\vec{S}_1N_6(n, m) \stackrel{2}{\prec} CylN_4(n, m)$ . ■

## Conclusion

This paper proposed two graph transformations: splitting and merging. Both can be used for simulating graph automata in order to make local transformations on the neighborhood. When used on regular graphs, the results we obtain can be compared with those for cellular automata on Cayley graphs. In particular, they can be used in this formalism and, conversely, local transformations on Cayley graphs can be applied to graph automata. We have given some examples of this kind in the paper. Only Lemma 4.5 cannot be related to cellular automata on Cayley graph since the PGA on a cylinder we have built for the simulation is not regular. The results of this paper can be combined with results from the Cayley graph approach to give other results as, for instance, one can replace  $N_6$  by  $N_3$  in Theorem 4.6. This study of the relationships –and differences– between both approaches would be interesting to pursue.

## References

- [1] D. Beauquier and M. Nivat. Tiling the plane with one tile. In *Symposium on Computational Geometry*, pages 128–138, 1990.
- [2] C. Berge. *Graphes*. Gauthier Villars, third edition, 1983.
- [3] M. Henle. *A combinatorial introduction to topology*. Dover Publication, 1979.
- [4] K. Kuratowski. *Introduction à la théorie des ensembles et à la topologie*. Institut de Mathématiques de l'Université de Genève, 1966.
- [5] B. Martin. Embedding torus automata into a ring of automata. *Int. Journal of Found. of Comput. Sc.*, 8(4):425–431, 1997.
- [6] B. Martin. A simulation of cellular automata on hexagons by cellular automata on rings. *Theoretical Computer Science*, 265:231–234, 2001.
- [7] B. Martin and C. Peyrat. A single-copy minimal-time simulation of a torus of automata by a ring of automata. *Discrete Applied Math.*, 155:2130–2139, 2007.
- [8] K. Morita and M. Harao. Computation universality of one-dimensional reversible (injective) cellular automata. *Trans. IEICE Japan*, E(72):758–762, 1989.
- [9] K. Morita, M. Margenstern, and K. Imai. Universality of reversible hexagonal cellular automata. In *Workshop on Frontiers between Decidability and Undecidability*, Brno, 1998.
- [10] 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.
- [11] Zs. Róka. *Automates cellulaires sur graphes de Cayley*. PhD thesis, École Normale Supérieure de Lyon, 1994.
- [12] Zs. Róka. One-way cellular automata on Cayley graphs. *Theoretical Computer Science*, 132(1–2):259–290, 1994.
- [13] Zs. Róka. Simulations between cellular automata on Cayley graphs. *Theoretical Computer Science*, 225:81–111, 1999.