Hyperbolic Recognition by Cellular Automata

Christophe Papazian, Eric Rémila

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON
n° 8512

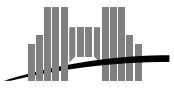
CNRS CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Hyperbolic Recognition by Cellular Automata

Christophe Papazian
Eric Rémila

Janvier 2002

Research Report N° 2002-03



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

INRIA

Hyperbolic Recognition by Cellular Automata

Christophe Papazian
Eric Rémila

Janvier 2002

Abstract

Graph automata were first introduced by P. Rosenstiehl. A. Wu and A. Rosenfeld showed later how a graph automaton can study its own structure, by building a system of signals that explore the underlying graph, giving in this way algorithms in linear time allowing to know if the graph is a regular grid. Then, E. Rémila extended this result to other geometrical structures.

We show here a very general method that allows to recognize all finite subsets of some Cayley graphs, without using some particular Euclidean information, like orientation, but some more general properties of automatic groups. Depending on the class of graphs we want to recognize, we can finally do different processing on the border of the detected geometrical structure, by using the small cancellations theorem. We study such graphs due to their good properties in network computing.

Keywords: Finite automata, Hyperbolic graphes

Résumé

Les graphes d'automates ont été introduits par P. Rosensthiel. A. Wu et A. Rosenfeld ont ensuite montré comment un graphe d'automates peut étudier sa propre structure, en construisant un système de signaux qui explorent le graphe sous-jacent, donnant ainsi un algorithme en temps linéaire permettant à un graphe d'automates de savoir si son graphe est un rectangle, puis E. Rémila a étendu cette reconnaissance à d'autres structures géométriques,

Nous proposons ici une méthode très générale qui permet de reconnaître les classes des sous-graphes finis de certains graphes de cayley, sans utiliser des propriétés euclidienne, comme la notion d'orientation, mais des propriétés très générales, comme celles des groupes automatiques. Selon la classe de graphe que l'on souhaite reconnaître, on peut alors effectuer des traitements différents de reconnaissance sur les bords de la structure géométrique détectée, en utilisant le théorème des petites simplifications.

Mots-clés: Automates finis, Graphes hyperboliques

Hyperbolic Recognition by Cellular Automata

Christophe Papazian (Christophe.Papazian@ens-lyon.fr)
Eric Rémila (Eric.Remila@ens-lyon.fr)

Introduction

Graph automata were first introduced by P. Rosenstiehl [12], under the name of *intelligent graphs*, surely because a network of finite automata is able to know some properties about its own structure. Hence P. Rosenstiehl created some algorithms that find Eulerian paths or Hamiltonian cycles in those graphs, with the condition that every vertex has a fixed degree [13]. Those algorithms are called "myopic" since each automaton has only the knowledge of the state of its immediate neighborhood. This hypothesis seems to be absolutely essential for the modelisation of the physical reality. A. Wu and A. Rosenfeld ([14] [15]) developed ideas of P. Rosenstiehl, using a simpler and more general formalism : the d-graphs. For each of these authors, a graph automata is formed by synchronous finite automata exchanging information according to an adjacent graph.

A. Wu and A. Rosenfeld have been especially interested in the problem of graph recognition. What is the power of knowledge of a parallel machine about its own architecture ? The motivation for this problem in parallelism is clear (can we control the structure of a parallel computer just using this computer ?) but this problem can also be interpreted in Biology or Physics as study of dynamic systems with local rules, or as a theoretical model of computer or mobile networks. In a general point of view, this is also a method to study the local algorithms on regular structures.

There is a general difficulty of this task, due to the finite nature of the automaton. As it has a finite memory, an arbitrary large number can not be stored in such a memory or in the signals exchanged by automata, . Hence, for example, coordinates can not be stored. Thus appropriate techniques have to be used : we use time differences to compute coordinates, and we will define signal flows to do comparisons between coordinates.

The first non-trivial result on this framework is due to A. Wu and A. Rosenfeld, who gave a linear algorithm allowing a graph automata to know if its graph is a rectangle or not. Then, E. Rémila [10] extended this result to other geometrical structures, with a system of cutting along certain lines, by using some methods of signal transmission in a fixed direction. In a recent paper [7], we gave a very general method that allows to recognize a large class of finite subgraphs of $\mathbb{Z}^2$ (and a lot of subclasses like classes of convex figures, of cell compatible figures ...) by putting orientations on edges and computing coordinates for each vertex.

In the present paper, we are interested in the problem of recognition of finite subgraphs of infinite regular hyperbolic networks. A new difficulty arises, due to the hyperbolic structure which is not intuitive, in our opinion : working in a hyperbolic plane is not as natural a working in $\mathbb{Z}^2$. Thus, we limit ourselves to networks constructed from Dehn groups (i. e. fundamental groups of compact orientable 2-manifolds, see [1], [2]), which are natural hyperbolic generalizations of the planar grid. Thus, the group structure can be used. Moreover, these groups are said *automatic* (in the sense of [6]), and the automatic structure gives us a powerful tool for our study.

The main results are a linear time (linear with the size of the studied graph) algorithm of recognition for a very large and natural class of finite subgraphs of hyperbolic regular networks and a quadratic time algorithm for the most general case.

1 Definitions

Notation : We will use a, b, c for constant on letters, u, v, w for words (and ϵ for the empty word), and x, y, z for variables on letters. We will use d, i, j, k for integers, e, f for edges, and μ, ν for vertices.

1.1 d-graph

A *graph* is a set $G = (V, E)$ with V a finite set of vertices, and E a subset of V^2, whose elements are called "edges". We will only consider graphs without loops (no (ν, ν) edges) and symmetric (if (μ, ν) is in E, so is (ν, μ)). A vertex μ is *a neighbor* of another vertex ν if (ν, μ) is in E. The *degree* of a vertex is the number of its neighbors; the degree of the graph is the maximum degree among its vertices. A vertex of degree d is called a d-vertex. A *path* of G from ν to μ is a finite sequence $(\nu_0, \nu_1, \ldots, \nu_p)$ (with $\nu = \nu_0$ and $\nu_p = \mu$) such that two consecutive vertices of the sequence are neighbors. A *cycle* is a path such that $\mu = \nu$. We say that G is connected if for each pair (ν, μ) of V^2, there exists a path of G from ν to μ. We will only consider connected graphs.

A *cyclic conjugate* of a word $w = x_1 ... x_n$ is a word $w_i = x_i ... x_n x_1 ... x_{i-1}$ for all $1 \leq i \leq n$ (see [2]).

A *labelling* λ is a mapping from E to a finite alphabet Σ. Given any path of G, the labelling of this path is the word built from the sequence of labels of successive edges of the path. Given any cycle of G, a word w is a *contour word* of this cycle iff w or w^{-1} is a cyclic conjugate of the labelling of this cycle.

Let d be a fixed integer such that $d \geq 2$. A d-graph is a 3-tuple $(G, \nu_0, \mathfrak{h})$, where $G = (V, E)$ is a symmetric connected graph with only two kinds of vertices : vertices of degree 1 (which are called #-vertices by Rosenfeld) and vertices of degree d, ν_0 is a d-vertex of G (which is called the leader, or the general), $\mathfrak{h}$ is a mapping from E to $\{1, 2, ..., d\}$ such that, for each d-vertex ν of V, the partial mapping $\mathfrak{h}(\nu, .)$ is bijective (injective if we do not consider the #-vertices). The subgraph G' of G induced by the set of d-vertices is called the underlying graph of G (see figure 1).

From any graph G' of degree at most d, we can construct a d-graph $(G, \nu_0, \mathfrak{h})$ whose underlying graph is G' : we add some #-vertices until the degree of each (non #) vertex is d, arbitrarily choose a leader and, for each vertex, build an order on its neighbors.

For each vertex, we call an up-edge, an edge that links this vertex to another vertex that is closer to the leader than itself. We call down-edge in the opposite case.

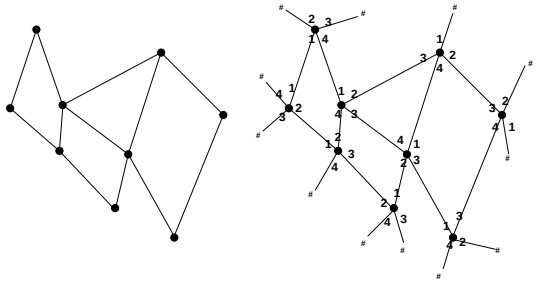


Figure 1: A graph and its associated 4-graph

1.2 Graph automata

1.2.1 Definition

A *finite d-automaton* is a pair (Q, δ) such that Q is a finite set of states with # in Q, and δ is a function from $Q \times (\mathbb{Z}_d)^d \times Q^d$ to Q such that, for each element Υ of $Q \times (\mathbb{Z}_d)^d \times Q^d$, $\delta(\Upsilon) = \#$ if and only if the first component of Υ is #.

A *graph automaton* is a set $M = (G_d, A)$ with G_d a d-graph, and A a finite d-automaton (Q, δ).

A *configuration* C of M is a mapping from the set of vertices of G_d to the set of states Q, such that $C(\nu) = \#$ iff ν is a #-vertex.

We define, for each vertex ν, the *neighborhood vector* $H(\nu) \in {\mathbb{Z}_d}^d$ in the following way : for each integer i of $\{1, 2, \ldots, d\}$, let μ_i denote the vertex such that $\mathfrak{h}(\nu, \mu_i) = i$. The i^{th} component of $H(\nu)$ is $\mathfrak{h}(\mu_i, \nu)$.

We compute a new configuration from a previous one by applying the transition function δ simultaneously to each vertex of G_d, computing a new state for each (non-#) vertex by reading its state and the states of its neighbors, using the vector of neighborhood :

$$\mathcal{C}_{new}(\nu) = \delta(\mathcal{C}(\nu), H(\nu), \mathcal{C}(\mu_1), ...\mathcal{C}(\mu_d))$$

(#-vertices always stay in the # state). Hence, we have a synchronous model, with local finite memory. The use of the neighborhood vector is necessary for this computation (see [11] for details).

There is a special state q_0 in Q that we call the *quiescent state*. If a vertex ν and all its (non-#) neighbors are in this state, ν will stay in this same state in the next configuration.

The *initial configuration* (the one from which we will begin the computation) is a configuration where all vertices are in the state q_0, except the leader, which is in a state q_{init}.

We say that a configuration $\mathcal{C}$ of G_d can be *reached* in M if $\mathcal{C}$ is obtained from the initial configuration after a finite number of steps of computations described above.

1.2.2 Similarity with a network

To simplify our explanations, we will consider that each automaton sends and receives signals to/from its neighbors at each step of computation, instead of looking at their entire states. It is easily possible, considering their states as sets of $d+1$ memory registers, one for keeping tracks about its own characteristics, and d others that contain messages for the neighbors, one register for each neighbor. Hence, at each step of computation, a vertex only looks at the messages computed for itself in the state of its neighbors.

To allow simultaneous exchanges, we consider a message as *a set of signals.*

1.3 The problem of recognition

Now, we will only use automata with two particular states : the accepting state and the reject state.

Definition 1 The class of accepted graphs for one d-automaton A *is the class of d-graphs G_d such that there is a configuration, where the leader is in the accepting state, that can be reached in (G_d, A).*

In the same way, we can define the class of rejected graphs.

Definition 2 A recognizer *is an automaton that rejects all d-graphs that it does not accept.*

In fact, we have a decision problem : As input, we have graphs, and for each automaton, we can compute if the graph is accepted or rejected. This natural problem has been formalized by A. Wu and A. Rosenfeld.

So the aim of the recognition of d-graphs is to build a recognizer for a given class of d-graphs.

1.4 Hyperbolic regular network

We call *a cell*, a cycle of a graph such that it is not possible to extract a shorter cycle of the graph from the cycle ("no vertex nor edges inside").Two different cells are said *adjacent* if they share one edge. A finite subgraph $G = (V, E)$ is said *cell compatible* if E is a union of cells. If, moreover, for each pair (C, C') of cells included in E with a common vertex ν, either C and C' are adjacent, or there exists cells $(C = C_0, C_1, ..., C_i = C')(i < d)$ included in E, such that each C_i contains the vertex ν and C_i is adjacent to C_{i+1}, then we say that G is *locally cell connected.*

Definition 3 *The* cell-neighborhood *of a vertex ν of a d-graph $G = (V, E)$ of $\Gamma(k, d)$ is the sub-graph of G composed by the cells containing the vertex ν.*

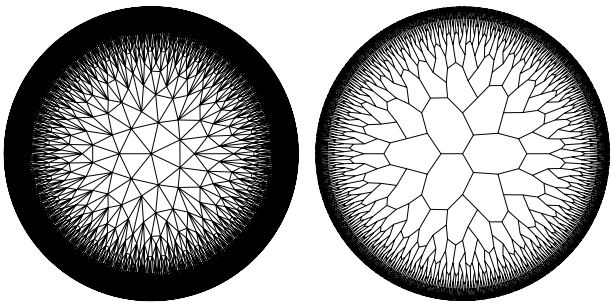


Figure 2: $\Gamma(3,7)$ and $\Gamma(8,3)$. $\Gamma(8,8)$ are not easily representable, due to a too large curvature.

A free group $\mathcal{G} =< S >=< S;\emptyset >$ (with S a finite set) is the set of finite words of Σ^* with the natural reduction $xx^{-1} = \epsilon$, where $\Sigma = S \cup S^{-1}$. A group $\mathcal{G} =< S; R >$, is the quotient group $< S > /d(< R >)$, with $d(< R >)$, the group generated by the set of words xrx^{-1} ($x \in S$, $r \in R$). Informally, G is the quotient group of $< S >$ where all words of R (and the symmetric words of words of R and the cyclic permutations of words of R) are equal to ϵ, the empty word. So we can have two different words of Σ^* that will represent a same element of $< S; R >$ (for example any element of R^* is equal to ϵ).

We will now consider, for $g \geq 2$, the alphabet $\Sigma_g = \{a_1, ..., a_{2g}, a_1^{-1}, ..., a_{2g}^{-1}\}$, the fundamental group (of the orientable compact 2-manifold of genus g) $\mathcal{G}_g =< \Sigma_g \ ; \ w_\perp = a_1 a_2 ... a_{2g} a_1^{-1} a_2^{-1} ... a_{2g}^{-1} >$ and the Cayley graph associated to such a group. Remember that a Cayley graph $G = (V, E)$ of a given group $\mathcal{G} =< S, R >$ is a graph such that V is the set of elements of $\mathcal{G}$ and $(v, w) \in E$ if and only if there is a generator $a \in S \cup S^{-1}$ such that va and w are the same element of $\mathcal{G}$. The symmetric unlabeled graph induced by the Cayley graph of $\mathcal{G}_g$ is $\Gamma(4g, 4g)$ defined as follows :

Definition 4 *A network $\Gamma(k,d)$ is the unique undirected planar graph, such that every cell has k edges, every vertex has d neighbors, and $\Gamma(k,d)$ can be drawn in such a way that each bounded region contains a finite number of vertices. It corresponds to the regular tessellations of Euclidean or hyperbolic planes.*

So, $\Gamma(4,4)$ is the infinite grid isomorphic to $\mathbb{Z}^2$, and $\Gamma(4g, 4g)$ is a natural extension in hyperbolic geometries.

Hence, the cycles of length $4g$ of $\Gamma(4g, 4g)$ are cells. A canonical labelling of $\Gamma(4g, 4g)$ is a labelling λ from the set of edges to Σ_g such that the cells admit $w_\perp$ as a contour word, and for all edges (ν, μ) and (ν, μ') we have $(\lambda(\nu, \mu) = \lambda(\nu, \mu')) \Rightarrow (\mu = \mu')$, and we have $\lambda(\nu, \mu) = \lambda(\mu, \nu)^{-1}$.

A finite graph G is called *a manifold on $\mathcal{G}_g$* if we have a valid labelling λ on G to Σ_g such that every cell of G admits $w_\perp$ as a contour word, every vertex does not have two outgoing edges with the same label, and we have $\lambda(\nu, \mu) = \lambda(\mu, \nu)^{-1}$.

As we can see from figure 3, the definition of locally cell connected graph forces the local structure of the cell-neighborhood of each vertex, and, conversely,

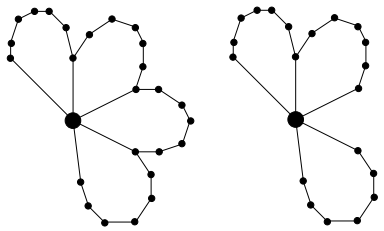


Figure 3: One possible structure of local cell-neighborhood for $\Gamma(8,8)$ on the left. On the right, a cell-neighborhood not allowed in locally cell connected manifold.

each connected subgraph of $\Gamma(4g,4g)$, such that the cell-neighborhood of each vertex is like the figure 3 (on the left), is locally cell connected.

Note that two isomorphic locally cell connected subgraphs of $\Gamma(4g,4g)$ also are isometric (from an Hyperbolic point of view). This is not true for all cell compatible subgraphs of $\Gamma(4g,4g)$. There are some kneecap effects around vertices that induce several possible local labellings (see figure 4).

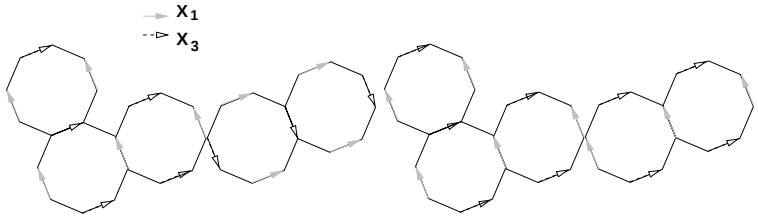


Figure 4: Two possible labellings of the same graph, due to the kneecap effect.

Each subgraph of $\Gamma(4g,4g)$ obviously admits a valid labelling, but some other graphs (for example : tori, cylinders, spheres ...) also do. Informally, we can say that each graph accepting a valid labeling can be locally mapped to the graph $\Gamma(4g,4g)$ and, moreover, all the local mappings have a kind of weak local coherence.

Moreover, a labelling of a locally cell connected subgraph of $\Gamma(4g,4g)$ is completely determined by the labels of the edges outgoing from a fixed vertex. This is not true for (non-locally) connected cell compatible subgraphs of $\Gamma(4g,4g)$. This fact is important for the determination of a possible isomorphism from a given graph to a subgraph of $\Gamma(4g,4g)$. It is the reason why we limit ourselves in studying the recognition of the class of $4g$-graphs whose underlying graph is isomorphic to a locally cell connected subgraph.

Finally, we say that a manifold $G=(V,E)$ (with a labelling λ) is mappable if a morphism φ exists from V onto the vertices of $\Gamma(4g,4g)$ such that the labelling is unchanged : $\lambda(\varphi(\nu),(\varphi(\mu)) = \lambda(\nu,\mu)$. Note that if φ is injective, then G is isomorphic to a subgraph of $\Gamma(4g,4g)$.

2 The Automaton

Now, we will construct a $4g$-automaton A_{4g} that recognizes the class of $4g$-graphs whose underlying graph is a cell locally connected figure of $\Gamma(4g, 4g)$. The process of recognition is divided into three steps :

First, each automaton tries to put coherent labellings on its edges. By doing this process, we recognize the locally cell connected manifolds of $\mathcal{G}_g$.

Afterwards, it (informally) puts words of the finite group $\mathcal{G}_g$ on its vertices, that must be coherent with the previously chosen labelling. Hence, we have an injection from the vertices of the graph to the elements of the group. By doing this second process, we recognize the mappable locally cell connected manifolds of $\mathcal{G}_g$.

Finally, automata do some processing on the border to detect the possible contradictions. By doing this last process, we recognize the subgraphs of $\Gamma(4g, 4g)$.

We will often say that *a vertex* sends a signal or computes a result to mean that *the copy of the automaton on this vertex* does such a task.

2.1 First Step : Labelling of edges

The process of labelling edges consists in exploring the potential $2g$-neighborhood of each $4g$-vertex of the underlying graph and giving labels to its edges. Of course, the labelling built by this process is a valid labelling.

The process is done by successive $4g+1$ time steps stages. During the first stage, indexes are given to the edges of the leader vertex. At the i^{th} stage, indexes are given to the edges of the i^{th} vertex. We first build a (depth) search tree in the graph, to have a natural order on the different vertices. Such a tree can easily be constructed by a graph automaton, see [14] and [15] for details.

Each stage is computed in $4g+1$ time steps using signals exploring the $2g$-neighborhoods of vertices.

$t = 0$, **Beginning**

The vertex ν sends a signal A_i^1 with $1 \leq i \leq 4g$ through each of its edges e_i (we use $\mathfrak{h}(\nu, .)$ to enumerate the edges of ν).

$1 \leq t < 2g$ **Transmission**

Each time a vertex receives a signal A_i^k $(k < 2g)$ through one of its edge, it sends a signal A_i^{k+1} through all of its other edges.

$t = 2g$ **Identification of cells**

When a vertex receives two signals A_i^{2g} by the edge e and A_j^{2j} by the edge f, it sends back a signal $B_{i,j}$ through e and $B_{j,i}$ through f.

$2g < t < 4g$ **Back transmission**

When a vertex receives a signal $B_{i,j}$, it sends it back through the edge from it previously received a signal A_i^k.

$t = 4g$, **Final computation**

The vertex ν must receive one or two signal $B_{i,j}$ (and $B_{i,j'}$) through its edge e_i. Informally, it allows to know that a cell of the $2g$-neighborhood exists between the edges e_i and e_j (and e_i and e'_j). As we know the neighbor edges

of any given edge, we can put labels on all the edges of ν, knowing at least one label previously given in a precedent stage.

$t = 4g + 1$, **Next step**

We send a signal to the next vertex ν', to indicate that ν' must begin the labelling process and to give the label of the edge (ν', ν).

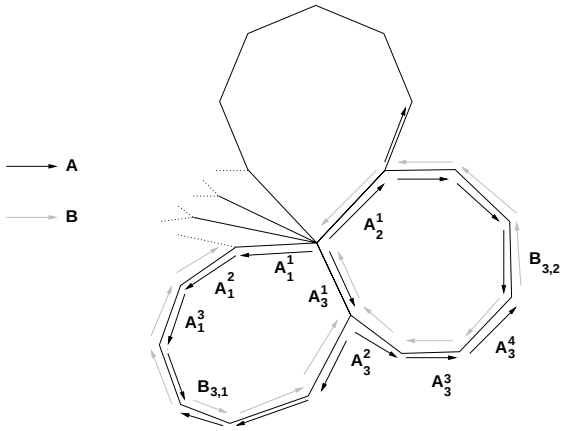


Figure 5: Exploration of the $2g$-neighborhood

Remark 1 *During this process, each vertex can moreover store, in its own state, the number of its border edges, i. e. edges which belong to only one (potential) cell. Each vertex has 0 or 2 border edges issued from itself due to local cell connectivity. Hence, the set of (undirected) border edges is a set of cycles.*

This first step recognizes the locally cell connected manifold of $\mathcal{G}_g$ in $\mathcal{O}(g|V| + r)$ time steps and build a valid labelling for the manifolds recognized.

2.2 Second Step : Coordinates of vertices

It is the main part of the work that must be done by the recognizer.

Now, as all edges are labeled, we would like to place coordinates from $\mathcal{G}$ on each vertex. Each finite automaton has a fixed memory, thus words of Σ_g^* of arbitrarily large length cannot be stored in such an automaton. So we will encode those coordinates into the time-space diagram of the automata, using a particular process.

In the Euclidean networks, we used coordinates of the form $a^n b^m$, in fact as words of $\{ab\}^*$, to represent the elements of the group $< a, b \mid aba^{-1}b^{-1} >= \mathbb{Z}^2$. In the hyperbolic problem, we will use coordinates of Σ_g^* to represent the elements of $\Gamma(4g, 4g)$

The coordinates (the words) have to be consistent with the labelling of edges. If we assign the word w_1 to the vertex ν_1 and the word w_2 to the neighbor vertex ν_2, and the label of (ν_1, ν_2) is a, then $w_1 a = w_2$ in $\mathcal{G}$. As we need to know if

the words we put on vertices are coherent with the labelling, we have to solve the word problem : given any two words w and w', are they equal ?

The word problem on hyperbolic groups : Building of a Tree

We explain here a new method to solve the word problem on hyperbolic Cayley graph, using finite automata. Remember that we could easily solve the word problem on Euclidean groups, by using Euclidean coordinates (that is the consequence of the Abelian properties of these groups). This is not the case for hyperbolic groups.

To solve the word problem on $\mathcal{G}_g =< \Sigma_g, w_\perp >\sim \Gamma(4g, 4g)$, we build a sublanguage L_T (a set of words) of S^* in bijection with $\mathcal{G}$, such that any words w of S^* could be easily transformed into the word of L_T that represents the same element of $\mathcal{G}$. To compare two given words consists now to transform them into the corresponding words of L_T and verify they are the same word.

Now, we explain how to build T, the graph of L_T. Given a $\Gamma(4g, 4g)$ and a given vertex $\odot$ (the origin) in this network, we can remark that every cell has one vertex that is closest to $\odot$ (distance n) and one vertex that is farthest from $\odot$ (distance $n + 2g$). To each cell C, we can associate a unique word w_C that is a cyclic conjugate of $w_\perp = a_1 a_2 ... a_{2g} a_1^{-1} ... a_{2g}^{-1}$ and w_C describes the successive labels of the edges in a contour word of the cell beginning at the closest vertex to $\odot$.

As we only take cyclic conjugates of w_0, and not words obtained by order reversing, words w_C induce an orientation of cells which gives a global orientation of Γ, as we can see on the figure 6.

Hence, for each cell C, the word $w_C = x_1 x_2 ... x_{4g}$ implies a natural numbering of the edges of the cell, from the first edge that is labelled with x_1 to the $4g^{th}$ edge that is labelled with x_{4g}. The first edge and the last edge are connected to the vertex of the cell C that is the closest to $\odot$. The edges labelled with x_{2g} and x_{2g+1} are connected to the farthest (from $\odot$) vertex of C.

Theorem 1 *If we consider the graph $\Gamma(4g, 4g)$ and the given vertex $\odot$, and, for each cell, we remove the $2g + 1^{th}$ edge, we transform the graph into a tree T, without changing the distance (from $\odot$) on the network.*

Proof The proof is easy by induction. Consider the cells that contain $\odot$. By removing one of the farthest edge of each of this cell, we do not change the distance between a vertex of these cells and $\odot$. So, the theorem holds for vertex at distance 1 from $\odot$.

Suppose, by induction hypothesis, that this is true for all vertices at distance n from $\odot$. All vertices at distance $n + 1$ was connected to vertex at distance n. If we do not removed an edge from a vertex, its distance is not changed. But if we remove one edge, it means that this vertex was connected to two distinct vertices at distance n (from $\odot$). Now, it is connected to only one of this two vertex. So we do not change the distance (from $\odot$) for all these vertices : the induction hypothetis is verified for $n + 1$.

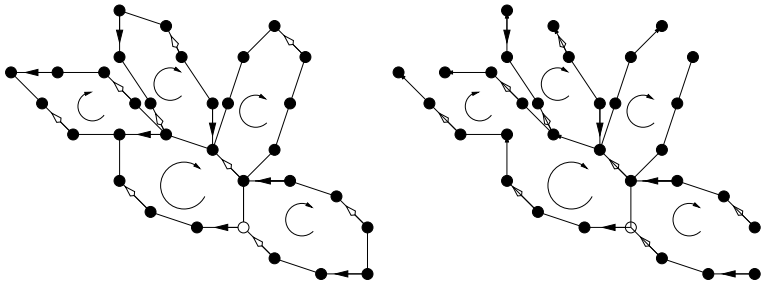


Figure 6: A part of $\Gamma(8,8)$, with labelling on the left, and the tree that we obtained on the right.

And the subnetwork obtained is a tree, as we can see that, for each vertex ν, there is only one neighbor of ν that is closer to $\odot$ than ν. It is a consequence of the fact that, in any infinite $\Gamma(k,d)$, given two separate geodesics (they do not share any edge) from ν to $\odot$, the first edge of the first geodesic is on the same cell that the first edge of the second one : the cell where ν is the farthest vertex (from $\odot$). But in T, we remove one of these two edges, so we can not have such a configuration : there is only one path from any ν to $\odot$ (see [8] for structural details on $\Gamma(k,d)$). □

Using the Tree to compute coordinates words

So, the tree T that we build implies a language L_T of geodesic words. We will use this language to compute the geodesic words that we will use as coordinates.

If we have a word u of L_T and consider the word ub ($b \in \Sigma_g$), the word ub is in L_T except if this is the obvious case $u = vb^{-1}$ (that implies $ub = v \in L_T$) or if b corresponds to a forbidden (removed in T) edge. In this case, the subword composed by the $2g$ last letters of ub correspond to a half cell of the form $x_1x_2...x_{2g} = w_{end}x_{2g}$ such that $w_{end}x_{2g}$ is a subword of a cyclic conjugate of $\overline{w_\perp} = a_{2g}...a_2a_1a_{2g}^{-1}...a_1^{-1}$. We have to change it into into the other half of the same cell corresponding to the word $x_{2g}x_{2g-1}...x_1 = x_{2g}\overline{w_{end}}$.

By this substitution, we transform $u_0 = ub = u'w_{end}x_{2g}$ into the word $u_1 = u'x_{2g}\overline{w_{end}} = u'b\overline{w_{end}}$. A new subword w_{half}, of length $2g$, corresponding to another forbidden half-cell, can be created in $u'x_{2g}\overline{w_{end}}$, but the only possibility is that the last letter of w_{half} is $x_{2g} = b$, since $x_{i+1}x_i$ cannot be a subword of a cyclic conjugate of $\underline{w_\perp}$. This enforces : $w_{half} = w_{end}x_{2g}$.

So if we can state $u' = u''w_{end}$, then we transform $u_1 = u'x_{2g}\overline{w_{end}}$ into $u_2 = u''x_{2g}(\overline{w_{end}})^2$.

Repeating this process until there is no possible substitution, a word $u_{last} = u'''x_{2g}(\overline{w_{end}})^j$ of L_T is obtained, in $\mathcal{O}$(length of w) , such that ub and u_{last} represent the same element of G_g.

Hence, we have a linear time algorithm θ such that given a word u of L_T and a letter $b \in \Sigma_g$ computes the word $v = ub$ (in $\mathcal{G}_g$) such that $v \in L_T$.

If we consider a word $v = y_1...y_n$ of Σ^*, to find the corresponding word u into L_T, just apply this algorithm on each prefix of v : $u_0 = \epsilon$ and $u_i = \theta(u_{i-1}y_i)$.

Hence $u = u_n$.

Hence, we have an algorithm to compute the geodesics in quadratic time on one automaton. It will be linear on graph automata, due to the fact we have as many automata as the size of the words we need to transform.

The algorithm of coordinates on graph automata

Now, we will use the properties on L_T to compute coordinates for each vertex. These coordinates will be encoded in the space time of each vertex, using a sequence of signals L_{a_i} that represents the sequence of letters a_i that will be a word of L_T corresponding to a particular element of $\mathcal{G}_g$. The sequence of signals is finished by a signal E (for "End").

Initialization So the leader vertex sends signals L_x to each of its neighbors, where $x \in S$, is the label of the edge trough which the signal L is sent. The next time step, it sends a signal E to each of its neighbors.

Main process Intuitively, each vertex will receive one or two words as a sequence of signals L_{x_i}. If the vertex receives only one word, there is no problem. If it receives several words, the vertex has to verify that the words are the same after a possible transformation. If the two words are not equal, there is a rejection.

When a vertex receives a word, letters of the word are stacked on a buffer of size $2g$.

But at any time, the vertex verifies if the buffer can be simplified, simulating the algorithm of computing geodesics of T. Hence, the words received by next vertices are always in T except the last letter. $2g$ time steps after receiving the first letter (signal L), and one time step before sending the first letter of the word to a particular neighbor, the considered vertex sends a signal L_x where x is the label of the edge that leads to this neighbor, to complete the word.

To finalize this process, each time a vertex does not have down-edges and is assigned one single element $\mathcal{G}$ by this process, it will send a "Final" signal through its up-edges. When a vertex received the "Final" signal through all its down-edges, it will send this signal through its up-edges. It is obvious that when the leader received this signal from all its edges, the process of coordinates succeeded.

This second step recognizes the localy cell connected mappable (for the labelling built by the first step) manifold of $\mathcal{G}_g$ in $\mathcal{O}(g.r)$ time steps, by building a morphism φ from the manifold to $\mathcal{G}_g$.

2.3 Final step : non-overlap verification

A mappable manifold is a subset of $\Gamma(4g, 4g)$ if and only if there are no two vertices with the same coordinates. So we have to verify the injectivity of φ. This is the last problem of *the overlaps*.

The general case of locally cell connected subgraphs of $\Gamma(4g, 4g)$

We have to compare the coordinates of each vertex to the coordinates of other vertices. The method is easy. We just build a search tree and for each vertex ν on the tree, we make the process of puting coordinates on other vertices, as if ν were the leader. If another vertex receives the empty word (in L_T), we know there is an overlap.

This process takes $\mathcal{O}(g.|V|.r)$ time steps. So, it is quadratic in the worst case. As we have to compare $\mathcal{O}(|V|^2)$ different coordinates of size $\mathcal{O}(r)$ with a space of size $\mathcal{O}(|V|)$, the time needed is at least $\mathcal{O}(|V|.r)$, using this method of resolution. We obtained $\mathcal{O}(r.\ln(r))$ for Euclidean networks, using the same arguments. Note that we can limit the verification to border vertices, but it does not actually change the complexity.

Theorem 2 *The class of locally cell connected subgraphs of* $\Gamma(4g, 4g)$ *can be recognized in quadratic time* $\mathcal{O}(g.|V|.r)$.

Linear time for the large subclass of convex figures

Now, we can carry out different processes on the border to find geometrical properties using linear time , especially the convexity property.

Definition 5 *A cell compatible subgraph, a mappable graph or a manifold* G *(of* $\Gamma(4g, 4g)$, $g > 1$ *) is* convex *iff there is only one single border cycle (the set of vertices of degree less than* $4g$ *defined a single connected cycle) and for any path* p *of* $2g + 1$ *consecutive vertices on the border such that* p *is a subsequence of a cell* c, c *is in* G.

This definition of convexity could be seen as *a local convexity property.* In fact, this definition of convexity that we give here is directly linked to the theorem of small cancellations ([4], for more details about this important theorem in hyperbolic groups). Due to this theorem (and more accurately, due to inherent property of hyperbolic networks), this local convexity implies strong consequences on graphs.

Theorem 3 (Theorem of small cancellations) *(Dehn 1911)*
For any $\mathcal{G}_g =< S, R >$, *if* w *is a word of* S^* *that is equal to* ϵ *(null word) in* $\mathcal{G}_g$, *then there exists a word* y, *subword of* w, *with* $|y| \leq 2g + 1$, *such that* y *can be reduced to* $y' = y$ *with* $|y'| < |y|$.

Theorem 4 *The class of convex cell connected subgraphs of* $\Gamma(4g, 4g)$ *can be recognized in linear time.*

In fact, it is very easy to verify such a property of convexity.To begin any process on the border, the leader begins a depth first search in the graph ([14]), until it finds a border vertex. Then, this vertex becomes a local leader on its border cycle. When the process on this cycle is finished, the depth first search resumes until it finds another border vertex on a cycle not yet reached or it

explored all the graph. Hence, the complexity of border algorithm is based on the number of vertices of the underlying graph. Then, just send a word trough the border cycle, like in the precedent part, and verify that the last $2g+1$ letters do not define a single cell. And if it is the case, verify that the cell is in the graph by looking at the other (non border) edges of the border vertices. Hence, we have a linear time algorithm to decide if a graph is convex.

And from the theorem of small cancellations, we can directly deduce the following lemma :

Lemma 1 *When a mappable graph is convex, there is no possible overlap : a mappable convex graph is a subgraph of* $\Gamma(4g, 4g)$

3 Extensions

The algorithm proposed here can be extended in many different ways. But we can see that we do not need any property of planarity in our algorithm, nor compass.

We give a solution for $\Gamma(4g, 4g)$, but this result is true for all $\Gamma(k, d)$ with k and d enough large to have the property of hyperbolicity.

The locally cell connected property is not necessary for linear time complexity : we can use a global cell connected property, with more state in the automaton. We must finally note that there is a trivial algorithm for non-cell connected graphs in exponential time.

References

[1] W. S. Massey, *Algebraic Topology, An introduction*, Springer, (1967)

[2] R. C. Lyndon, P. E. Schupp, *Combinatorial Group Theory*, Springer, (1977)

[3] M. Gromov, *Hyperbolic groups*, in S.M. Gersten (ed.), *Essays in Group Theory*, Mathematical Sciences Research Institute Publications 8, Springer Verlag, Berlin (1987), 75–263

[4] E. Ghys, P. de la Harpe *Sur les Groupes Hyperboliques d'après Mikhael Gromov* Progress in Mathematics, Birkhäuser, (1990)

[5] J.M Alonso, T. Brady, D. Cooper, V. Ferlini, M. Lustig, M. Mihalik, M. Shapiro and H. Short, *Notes on word hyperbolic groups*, in E. Ghys, A. Haefliger and A. Verjosky (eds.) *Group Theory from a Geometric Viewpoint*, World Scientific, Singapore (1991), 3-63

[6] David B. A. Epstein *Word Processing in Groups* Jones and Bartlett Publishers, (1992)

[7] C. Papazian, E. Rémila *Linear time recognizer for subsets of* Z^2 Proceedings of Fondamentals of Computation Theory (FCT), (2001), LNCS 2138, 400-403

[8] C.Papazian, E. Rémila *Some Properties of Hyperbolic Networks* Proceeding of Discrete Geometry for Computer Imagery (DGCI), (2000), LNCS 1953, 149-158

[9] J. Mazoyer, C. Nichitiu, E. Rémila, *Compass permits leader election*, Proceedings of Symposium on Discrete Algorithms (SODA), SIAM Editor (1999), 948-949

[10] E. Rémila, *Recognition of graphs by automata*, Theoretical Computer Science 136, (1994), 291-332

[11] E. Rémila, *An introduction to automata on graphs*, Cellular Automata, M. Delorme and J. Mazoyer (eds.), Kluwer Academic Publishers, Mathematics and Its Applications 460, (1999), 345-352

[12] P. Rosenstiehl, *Existence d'automates finis capables de s'accorder bien qu'arbitrairement connectés et nombreux*, Internat. Comp. Centre 5 (1966), 245-261

[13] P. Rosensthiel, J.R Fiksel and A. Holliger, *Intelligent graphs: Networks of finite automata capable of solving graph problems* , R. C. Reed, Graph Theory and computing, Academic Press, New-York, (1973), 210-265

[14] A. Wu, A. Rosenfeld, *Cellular graph automata I*, Information and Control 42 (1979) 305-329

[15] A. Wu, A. Rosenfeld, *Cellular graph automata II*, Information and Control 42 (1979) 330-353