Compact representation of triangulations

Luca Castelli Aleardi, Olivier Devillers, Gilles Schaeffer

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Compact representation of triangulations

Luca Castelli Aleardi — Olivier Devillers — Gilles Schaeffer

N° 5433

17 decembre 2004

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--5433--FR+ENG

Compact representation of triangulations

Luca Castelli Aleardi* , Olivier Devillers†, Gilles Schaeffer‡

Thème SYM — Systèmes symboliques
Projet Geometrica

Rapport de recherche n° 5433 — 17 decembre 2004 — 20 pages

Abstract:

We consider the problem of representing compact geometric data structures maintaining an efficient implementation of navigation operations. For the case of planar triangulations with m faces, we propose a compact representation of the connectivity information that improves to 2.175 bits per triangle the asymptotic amount of space required and that supports navigation between adjacent triangles in constant time. For triangulations with m faces of a surface with genus g, our representation requires asymptotically an extra amount of $36(g-1)\lg m$ bits. The structure also allows constant time access to vertex specific data, like coordinates, but the paper does not address the compression of this geometric information.

Key-words: graph encoding, succinct representations, triangulations, geometric data structures.

This work has been supported by the French "ACI Masses de données" program, via the Geocomp project: `http://www.lix.polytechnique.fr/Labo/Gilles.Schaeffer/GeoComp/`

* INRIA - Geometrica and LIX, Ecole Polytechnique, 91128 Palaiseau
† INRIA - Geometrica
‡ LIX, Ecole Polytechnique, 91128 Palaiseau

Représentation compacte de triangulations

Résumé :

Nous considérons le problème de représenter des structures géométriques de manière compacte en gardant une implémentation efficace des opérations de navigation. Pour le cas des triangulations planaires à m faces, nous proposons une représentation compacte de l'information combinatoire qui améliore à 2.175 bits par triangle le coût asymptotique en éspace et qui permet la navigation entre triangles adjacents en temps constant.

Pour les triangulations à m faces d'une surface de genre g, notre représentation nécessite asymptotiquement de $36(g-1)\lg m$ bits supplémentaires. La structure permet aussi l'accès en temps constant des informations associées aux sommets, notamment leurs coordonnées, cependant nous ne traitons pas ici la compression de cette information géométrique.

Mots-clés : Codage de graphes, représentations compactes, triangulations, structures de données géométriques.

1 Introduction

1.1 Contribution

The problem of representing compactly the connectivity information of a two-dimensional triangulation has already been addressed for compression purpose and a coder reaching the bound of 1.62 bits per triangle has been achieved for the class of triangulations with a triangular boundary [15]. In this paper, our purpose is not to compress the data for storage or network transmission, but to design a compact representation that can be used in main memory and supports navigation queries. More precisely, given a triangulation of m triangles, we propose a structure using $2.175m + O\left(m\frac{\lg\lg m}{\lg m}\right)$ bits and that supports access from a triangle to its neighbors in $O(1)$ worst case time.[1] This storage is asymptotically optimal for the class of planar triangulations with boundary.

Our approach extends directly to triangulations with m faces of a surface with genus g. In this case, the structure uses $2.175m + 36(g-1)\lg m + O\left(m\frac{\lg\lg m}{\lg m} + g\lg\lg m\right)$ bits, which remains asymptotically optimal for $g = o(m/\lg m)$. For $g = \Theta(m)$, we still have an explicit dominant term, which is of the same order as the cost of a pointer-based representation.

1.2 Related work on compact representations of graphs

There is relatively few previous work on that problem. Several people have looked for practical solutions from the programming point of view [10] improving by a constant factor usual representation [2]. More recently, with a different approach, Blandford *et. al.* [1] presented a compact representation for separable graphs that requires $O(n)$ bits and supports adjacency and degree queries in constant time.

On a more theoretical side, the seminal work of Jacobson [9] described how to represent planar graphs with $O(n)$ bits, allowing adjacency queries in $O(\lg n)$ time: this result is based on a four-pages decomposition of planar graphs. This approach has been improved by Munro and Raman [13]: using a succinct representation of balanced parenthesis words they show how to achieve $O(1)$ time for the implementation of adjacency between vertices and degree queries: for the case of planar triangulations with e edges and n vertices only $2e+8n$ bits are asymptotically required, that is, in terms of the number m of faces, between $7m$ and $12m$ bits depending on the boundary size.

The best known theoretical result is due to Chuang *et al.* [4]: using again a compact representation for multiple parentheses and exploiting properties of canonical orderings they improved the higher order term for the space to $2e+n$ for the case of triangulations (which is equivalent to $3.5m$ for triangulation with a triangular boundary), with slightly different navigation primitives than ours. This result has been further extended and improved for the general case of planar graphs by Chiang *et al.* [3].

[1] As discussed in Section 2.1, access to a word of $\lg m$ bits takes constant time.

1.3 Outline

The outline of this paper is as follows. In Section 2 we give some definitions and a general overview of our representation of triangulation. Sections 3 to 6 describe precisely the different data structures involved and study the amount of storage needed. Section 7 shows how to navigate between neighboring triangles. Finally, in Appendix 8 we analyze the construction time of our data structure and in Appendix 9, we compute the entropy of the class of planar triangulations with boundary.

1.4 Overview of our structure

As in previous strategy for succinct representation of combinatorial data structures ([14]) the main idea is to decompose the combinatorial structure into small sub-triangulations of poly-logarithmic size.

In our case we divide the initial triangulation in pieces (*small triangulations*) having $\Theta(\lg^2 m)$ triangles, and each small triangulation is then divided into *planar* sub-triangulations (*tiny triangulations*) of size $\Theta(\lg m)$.

Then we construct a three level structure. The first level is a graph linking the $\Theta\left(\frac{m}{\lg^2 m}\right)$ small triangulations, or more precisely a map, since the relative order of neighbors around a small triangulation matters. This map is classically represented with pointers of size $O(\lg m)$, and in view of its number of nodes, it uses a sub-linear storage. The second level consists in a map linking the tiny triangulations. The nodes of this map are grouped according to the small triangulation they belong. This allows to use local pointers of size $O(\lg \lg m)$ to store adjacencies between tiny triangulations. Vertices on the boundary of tiny triangulations are colored depending on the number of tiny triangulations they belongs. The combinatorial information of a tiny triangulation is not explicitly stored at this second level, we just store a pointer to the third level: the catalog of all possible tiny triangulations. The coloring of the boundary of a tiny triangulation is also encoded through a pointer to a catalog of all possible colorings. The whole size of all these pointers to the third level can be proved to be 2.175 bits per triangle and all other informations are sub-linear.

From a very general point of view, the basic framework has some similarity with previous work for trees [14], however the combinatorics of combining tiny triangulations into a big triangulation is significantly more involved. In particular the fact that we have to allow our (planar) tiny triangulations to have an arbitrarily complex boundary causes our representation to be optimal in space only for the class of planar triangulations with boundary: the dominant term in the asymptotic storage space, 2.175 bits per face, is indeed precisely the entropy of this class of triangulations.

2 Preliminaries

2.1 Model of computation

As in previous works about succinct representations of binary trees, our model of computation is a RAM machine. Under this model the access and arithmetic operations can be implemented in $O(1)$ time on words of size $\log_2 m$.

Any element in a memory word can be accessed in constant time, once we are given a pointer to a word and an integer index inside. The machine word size matches the problem size, in the sense that a word is large enough to store the input problem size. We use $\lg m$ to denote $\lceil \log_2 m + 1 \rceil$. From now on, when we will speak about time complexity of an algorithm we refer to the number of elementary operations on words of size $\lg m$, and while speaking about storage we always refer to the size of an object in term of bits.

2.2 Notations and vocabulary

We start by setting some notations that will be used in the rest of this work.

The initial triangulation is denoted $\mathcal{T}$ and its size m (from now on the size of any triangulation is its number of triangles). When the triangulation $\mathcal{T}$ is not planar, we denote by g its genus. The *small* triangulations, of size between $\frac{1}{3}\lg^2 m$ and $\lg^2 m$, are denoted $\mathcal{ST}_i$. Finally the *tiny* triangulations, of size between $\frac{1}{12}\lg m$ and $\frac{1}{4}\lg m$, are denoted $\mathcal{TT}_j$.

All tiny triangulations shall be planar triangulations with one boundary cycle. However as subtriangulations of $\mathcal{T}$, these tiny triangulations will share their boundary edges. More precisely a boundary edge can be shared by two different tiny triangulations or can also appear twice on the boundary of one tiny triangulation. A *side* of a tiny triangulation $\mathcal{TT}_j$ is a maximal set of consecutive boundary edges that are shared by $\mathcal{TT}_j$ with a same tiny triangulation $\mathcal{TT}_{j'}$ (possibly with $j' = j$). The boundary of a tiny triangulation is divided in this way in a cyclic sequence of sides.

The adjacencies between the small triangulations $\mathcal{ST}_i$ are stored in a graph denoted F, those between tiny triangulations $\mathcal{TT}_j$ in a graph G. The part of G corresponding to pieces of $\mathcal{ST}_i$ is denoted G_i. To be more precise F and G are *maps*: at each node the set of incident arcs is stored in an array, whose elements are sorted to reflect the circular arrangement of the sides of the triangulation.

The exhaustive set of all possible triangulations with at most $\frac{1}{4}\lg m$ triangles is stored in a structure denoted A while the set of all coloring of a boundary with less than $\frac{1}{4}\lg m$ vertices is stored in a structure called B.

As discussed more precisely in Appendix A, these maps can have loops (corresponding to self intersection of the boundary of a subtriangulation) and multiple arcs (corresponding to two subtriangulations sharing different sides). On the other hand, by construction, these maps will have only faces of degree at least 3 (because only one arc is set for each side).

For the sake of clarity, from now on we will use the word *arcs* and *nodes* to refer to edges and vertices of the maps F, G and G_i, and keep the word edges and vertices only for the edges and vertices of $\mathcal{T}$ and of the subtriangulations.

2.3 Operations on the triangulation

As explained in the previous section the aim of this work is to present a compact representation that allows the user to access and navigate efficiently in a triangulation: for this purpose we describe here the set of primitive operations that are supported by our representation in $O(1)$ time.
- $Triangle(v)$: returns a triangle incident to vertex v;
- $Index(\triangle, v)$: returns the index of vertex v in triangle $\triangle$;
- $Neighbor(\triangle, v)$: returns the triangle adjacent to $\triangle$ opposite to vertex v of $\triangle$;
- $Vertex(\triangle, i)$: returns the vertex of $\triangle$ of index i.

2.4 Previous data structures and algorithms used

Rank/Select structure. Let us consider a bit-vector $v = b_1 \dots b_p$ consisting of p bits and having weight (the number of 1) q. We would like to perform on v the following operations:
- $Rank1(i)$ (resp. $Rank0(i)$) returns the number of 1 (resp. 0) that precede b_i;
- $Select1(k)$ (resp. $Select0(k)$) returns the position of the k-th 1 (resp. 0) in v.

We need a structure able to answer these queries in constant time. This problem was addressed very much in detail in the literature and compact solutions have been proposed (see [17], [16], [5] and ref. therein). However since space is not critical at the point where we need a rank/select structure, we content ourselves with a simple implementation using $O(p \lg p)$ bits which we describe in Section 4.

Tree partitioning. The decomposition of the initial triangulation we will use is based on a partitioning procedure for ordered trees described in [7]. The main result we need is expressed by the following lemma:

Lemma 1. *Given a binary tree $\mathcal{B}$ on n nodes (each node has degree at most 3) and a positive integer parameter M, it is possible to produce in $O(n)$ time a partition of $\mathcal{B}$ into sub-trees $\mathcal{B}_i$, such that the size of every subtree satisfies $3M \geq \|\mathcal{B}_i\| \geq M$.*

3 Exhaustive list of all tiny triangulations

All possible triangulations having i triangles $i \leq \frac{1}{4} \lg m$ are generated and their explicit representations are stored in a table A. This table is sorted by increasing size of the triangulations, so that the bit size of an index is proportional to the size of the corresponding triangulation. In the rest of this section we describe the organization of the structure (see also Figure 1) and we analyse the storage. The construction of the structure in sub-linear time is given in Appendix 8.1.

3.1 Description of the representation

- A is a table of size $\frac{1}{4} \lg m$, in which the ith element is a pointer to Table A_i.

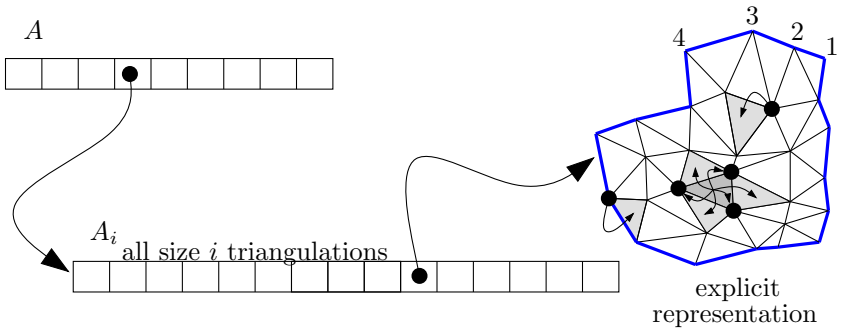

Figure 1: Storage of all tiny triangulations

• A_i is a table containing all possible triangulations having exactly i triangles. The jth element is a pointer to an explicit representation $A_{i,j}^{explicit}$ of the triangulation $A_{i,j}$.
• $A_{i,j}^{explicit}$ contains two fields:
— $A_{i,j}^{explicit}.vertices$ is the table of the vertices of $A_{i,j}$. Each vertex just contains the index of an incident triangle in Table $A_{i,j}^{explicit}.triangles$. By convention, the boundary vertices are first in that table, and stored in the counter-clockwise order of the boundary of $A_{i,j}$. For boundary vertices, the incident triangle stored is required to be the one incident to next edge on the boundary.
— $A_{i,j}^{explicit}.triangles$ is the table of the triangles of $A_{i,j}$. Each triangle contains the indices of its vertices in $A_{i,j}^{explicit}.vertices$ and of its neighbors in $A_{i,j}^{explicit}.triangles$. Triangles on the boundary have *null* neighbors.

3.2 Storage analysis

Lemma 2. *The storage of Table A, and of all the information associated with Tables A_i requires asymptotically $O(m^{0.55})$ bits.*

Proof. • A is a table of size $\frac{1}{4}\lg m$ of pointers of size $\lg m$ and thus costs $O(\lg^2 m)$.
• According to the enumeration results of Appendix 9, A_i is a table of $2^{2.175i} \leq 2^{2.175\frac{1}{4}\lg m}$ pointers on $\lg m$ bits, thus the storage of A_i requires less than $O(2^{2.175\frac{1}{4}\lg m}\lg m)$ bits.
• The explicit representation $A_{i,j}^{explicit}$:
— $A_{i,j}^{explicit}.vertices$ and $A_{i,j}^{explicit}.triangles$ are two tables of size less than $i \leq \lg m$. Each element consists in several indices of value less than i that are representable with $\lg\lg m$ bits.
Thus the whole size of one $A_{i,j}^{explicit}$ is $O(\lg m \lg\lg m)$ bits and the total size of all $A_{i,j}^{explicit}$ associated to A_i is less than $O(2^{2.175\frac{1}{4}\lg m}\lg m\lg\lg m)$ bits.

Finally the total size for the storage of all tiny triangulations is obtained by summing over the different values of i. This yields $O(2^{2.175\frac{1}{4}\lg m}\lg^2 m \lg\lg m) = O(m^{0.55})$. □

4 Rank/Select on bit-vectors

Later on, we will need to mark some vertices on the boundary of a tiny triangulation, and we will do this with the help of a bit vector. To be able to answer Rank/Select queries in constant time on these bit vectors, we use an exhaustive encoding of all bit-vectors of size p and weight q in a Table B. In the rest of this section we provide the description and analysis of the structure. Its construction in sub-linear time is given in Section 8.2.

4.1 Description of the representation

• B is a bi-dimensional array of size $\frac{1}{4}\lg m \times \frac{1}{4}\lg m$: each entry $B(p,q)$ is a pointer to Table B_{pq}.
• B_{pq} is a table containing for the kth bit-vector of size p and weight q a pointer to a structure B^{RS}_{pqk} allowing Rank/Select in constant time.
• B^{RS}_{pqk} is a table of length p with two fields storing the precomputed result for $Rank_1$ and $Select_1$:
  — $B^{RS}_{pqk}(i).rank$ is the number of '1' that precede the i-th bit.
  — $B^{RS}_{pqk}(i).select$ is the position of the i-th '1' in the vector.

4.2 Storage analysis

Lemma 3. *The storage of Table B, and of all the information associated with Tables B_{pqk} requires asymptotically $O(m^{\frac{1}{4}}\lg m \lg\lg m)$ bits.*

Proof. • B is a table of $(\frac{1}{4}\lg m)^2$ pointers of size $\lg m$, its size is $O(\lg^3 m)$ bits.
• B_{pq} is a table containing $\binom{p}{q}$ pointers of size $O(\lg m)$.
• $B^{RS}_{pqk}(i).rank$, $B^{RS}_{pqk}(i).select$ are all integers less than $\frac{1}{4}\lg m$ and then representable on $\lg\lg m$ bits. The size of B^{RS}_{pqk} is $O(\lg m \lg\lg m)$.

The total amount of space required for storing all the bit-vectors of size (and weight) less than $\frac{1}{4}\lg m$ is then $\sum_{p,q}\binom{p}{q}O(\lg m\lg\lg m) = \left(\sum_p 2^q\right)O(\lg m\lg\lg m)$, which is bounded by $2^{\frac{1}{4}\lg m+1}O(\lg m\lg\lg m) = O(m^{\frac{1}{4}}\lg m\lg\lg m))$. □

5 Map of tiny triangulations

The main triangulation is split into small triangulations which are themselves split into tiny triangulations. In this section we describe the map G that stores the incidences between tiny triangulations. The memory for this map is organized by gathering nodes of G that correspond to tiny triangulations that are part of the same small triangulation $\mathcal{ST}_i$ in a

submap G_i. The purpose of this partition is to allow for the use of local pointers of small size for references inside a given submap G_i. The map G may have multiple arcs or loops but all its faces have degree ≥ 3. Each arc of G between $\mathcal{TT}_j$ and $\mathcal{TT}_{j'}$ corresponds to a side shared by $\mathcal{TT}_j$ and $\mathcal{TT}_{j'}$.

A linear time construction of G is described in Section 8.3.

5.1 Description of the representation

The memory dedicated to G is organized in a sequence of variable size zones, each dedicated to a G_i. The memory requirements are analyzed afterward.

In the zone for G_i, for each node $G_{i,j}$ corresponding to a tiny triangulation $\mathcal{TT}_{i,j}$, we have the following informations:

— $G^t_{i,j}$ is the number of triangles in $\mathcal{TT}_{i,j}$.

— $G^b_{i,j}$ is the size of the boundary of $\mathcal{TT}_{i,j}$.

— $G^A_{i,j}$ is the index of the explicit representation of $\mathcal{TT}_{i,j}$ in Table $A_{G^t_{i,j}}$.

— $G^s_{i,j}$ is the degree of the node $G_{i,j}$ in the map G_i ($G^s_{i,j}$ is also the number of sides of $\mathcal{TT}_{i,j}$)

— $G^B_{i,j}$ is the index in Table $B_{G^b_{i,j},G^s_{i,j}}$ of a bit-vector of size $G^b_{i,j}$ and weight $G^s_{i,j}$. (This bit vector encodes the way the boundary of $\mathcal{TT}_{i,j}$ splits into sides: the ith bit is 0 if the ith vertex on the boundary of $\mathcal{TT}_{i,j}$ is inside a side, or 1 if this vertex separates two sides.)

— Each of the $G^s_{i,j}$ arcs of G_i that are incident to $G_{i,j}$ is described by some additional information (beware that loops appear twice). Assume that the kth such arc connects $G_{i,j}$ to a neighbor $G_{i',j'}$ in G, then we store:

— $G^{address}_{i,j,k}$ the relative address of the first bit concerning the node of the neighbor in the memory zone associated to its small triangulation $G_{i'}$.

— $G^{back}_{i,j,k}$ the index k' of the side corresponding to the current arc in the numbering of sides at the opposite node $G_{i',j'}$.

— $G^{small}_{i,j,k}$ the index of the small triangulation $G_{i'}$ in the table of the neighbors of G_i in the main map F (if $i' = i$ then this index is set to 0).

5.2 Storage analysis

Lemma 4. *The storage of map G requires asymptotically* $2.175m+O(g\lg\lg m)+O\left(m\frac{\lg\lg m}{\lg m}\right)$ *bits.*

Proof. For each node:

— $G^t_{i,j}$, $G^b_{i,j}$ and $G^s_{i,j}$ are less than $\frac{1}{4}\lg m$ and thus can be stored on $\lg\lg m$ bits each.

— $G^A_{i,j}$ is an index in $A_{G^t_{i,j}}$ whose size is bounded by $2^{2.175G^t_{i,j}}$ according to the result of Appendix B. It can thus be stored on $2.175G^t_{i,j}$ bits.

— $G^B_{i,j}$ is an index in $B_{G^b_{i,j},G^s_{i,j}}$ whose size is $\binom{G^b_{i,j}}{G^s_{i,j}} \leq {G^b_{i,j}}^{G^s_{i,j}}$ bits.

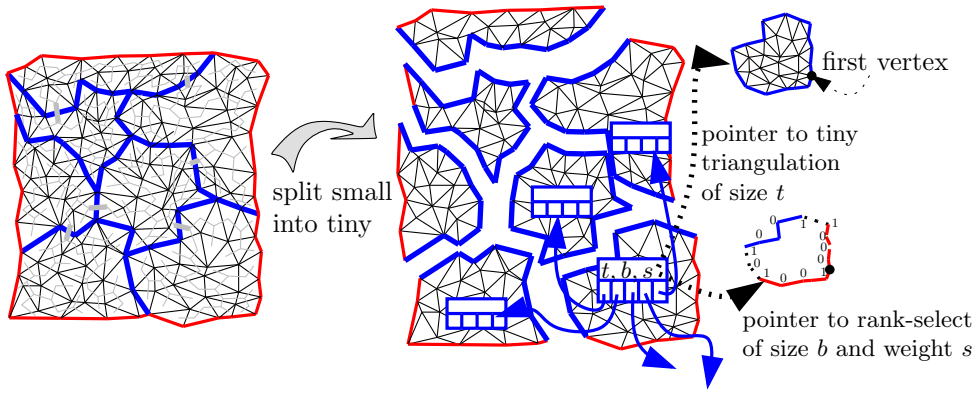

Figure 2: Submap G_i of a small triangulation

— The number of tiny triangulations neighboring $G_{i,j}$ is $G^s_{i,j} < \frac{1}{4}\lg m$. We have for each:

— the pointers $G^{address}_{i,j,k}$ are stored on $K \lg\lg m$ bits (K is determined below).

— $G^{back}_{i,j,k}$ is less than $\frac{1}{4}\lg m$ and thus can be stored on $\lg\lg m$ bits.

— $G^{small}_{i,j,k}$ requires $2\lg\lg m$ bits of storage: indeed a small triangulation has at most $\lg^2 m$ triangles, hence at most $\lg^2 m$ edges on its boundary, thus the table of the neighbors of G_i in F has less than $\lg^2 m$ entries.

Since each arc appears on at most two nodes, the cost per arc can be evaluated independently as $2(K+3)\lg\lg m$ bits per arc. It then remains for node $G_{i,j}$ of G_i a cost of $3\lg\lg m + 2.175 G^t_{i,j} + G^s_{i,j}\lg G^b_{i,j}$.

The number of nodes is at most $12\lg m$ and the number of arcs (including arcs directed to other $G_{i'}$) is bounded by the number of edges of $\mathcal{T}$ incident to triangles of $\mathcal{ST}_i$, that is by $\lg^2 m$.

The cost for G_i is thus $C_i \leq 2(K+3)\lg^2 m \lg\lg m + 12\lg m(3\lg\lg m + 2.175\frac{1}{4}\lg m + \frac{1}{4}\lg m\lg\lg m)$. Taking $K=5$, we have $\lg C_i < K\lg\lg m$ for all $m \geq 2$, which validates our hypothesis for the storage of $G^{address}_{i,j,k}$.

The overall cost for the complete map G is obtained by summing over i, j:

$$\sum_i \sum_j \left(3\lg\lg m + 2.175 G^t_{i,j} + G^s_{i,j}(\lg G^b_{i,j}) + G^s_{i,j}\cdot 8\lg\lg m\right)$$
$$\leq \quad 2.175\sum_{i,j} G^t_{i,j} + 9\lg\lg m \sum_{i,j} G^s_{i,j} + 3\lg\lg m \cdot 12\frac{m}{\lg m}.$$

The sum over $G^t_{i,j}$ is the total number of triangles, *i.e.* m. The sum over $G^s_{i,j}$ is the sum of the degrees of the nodes of the map G, or equivalently, twice its number of arcs.

Since G has only faces of degree ≥ 3, its number a of arcs linearly bounds its number f of faces: $2a = \sum_f d(f) \geq 3f$. Euler's formula can then be written (with n for the number of nodes and g for the genus of G which is also the genus of $\mathcal{T}$):

$$3(a+2) = 3n + 3f + 6g \qquad \Leftrightarrow \quad a \leq 3n + 6(g-1).$$

Finally the number n of nodes of G is bounded by $12m/\lg m$, so that the cost of representing G is

$$C = 2.175m + 9\lg\lg m \cdot 2\left(3 \cdot 12\frac{m}{\lg m} + 6(g-1)\right) + 3\lg\lg m \cdot 12\frac{m}{\lg m},$$

and the lemma follows. Observe also that the bound $g \leq \frac{1}{2}m + 1$ yields $\lg C \leq 2\lg m + 8$ for all m which will be used in the next section. □

6 Map of small triangulations

The last data structure needed is a map F that describes the adjacency relations between small triangulations. The map F has a genus less or equal to the genus of G and it contains no faces of degree less than 3. We adopt here an explicit pointer based representation. The linear time construction of F is discussed in Section 8.3.

6.1 Description of the representation

We store for each node of F its degree, a link to the corresponding part of G and the list of its neighbors. More precisely, for a node F_i corresponding to a small triangulation $\mathcal{ST}_i$:
• F_i^s is the degree of node F_i in the map F (it corresponds to the number of small triangulations adjacent to $\mathcal{ST}_i$);
• F_i^G a pointer to the submap G_i of G, associated to the small triangulation $\mathcal{ST}_i$.
• A table of pointers to neighbors: $F_{i,k}^{address}$ is the address of the kth neighbor of F_i in F.

6.2 Storage analysis

Lemma 5. *The storage of map F requires asymptotically* $36(g-1)\lg m + O\left(\frac{m}{\lg m}\right)$ *bits.*

Proof. Recall that a small triangulation contains between $\frac{1}{3}\lg^2 m$ and $\lg^2 m$ triangles, thus map F has at most $3m/\lg^2 m$ nodes.
• F_i^s is less than $\lg^2 m$, and thus representable on $2\lg\lg m$ bits.
• the address of G_i is a pointer of size bounded by $2\lg m + 8$.
• pointers to neighbors $F_{i,k}^{address}$ are stored on $K'\lg m$ bits each (K' chosen below).

Summing on all the small triangulations we obtain that the bit size of F is $(2\lg m + 8) \cdot 3m/\lg^2 m + K'\lg m \sum_i F_i^s$. The sum of the F_i^s is the sum of the degrees of nodes of F, which is also twice its number of arcs.

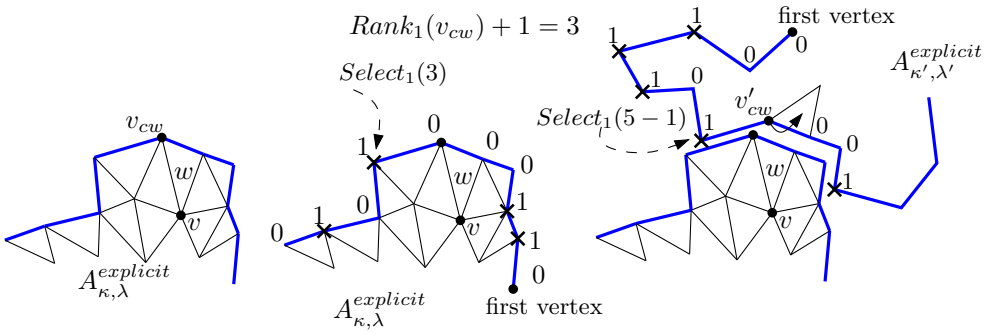

Figure 3: Going to the neighbor

In analogy with what was done for the map G, the number of arcs of F can be bounded more precisely by three times its number of nodes, which is less than $3m/\lg^2 m$, plus six times the genus minus one of F, which is bounded by the genus of $\mathcal{T}$. Using the bound $g < \frac{1}{2}m + 1$ on the genus, the value $K' = 3$ is seen to satisfy the constraints for $m \geq 5$. Finally the total bit cost for F is thus: $36(g-1)\lg m + O(m/\lg m)$. □

7 Navigation

Triangle and vertex representations In our structure, a triangle t is represented by a triple (F_i, a, w) where F_i is a node of F such that $t \in \mathcal{ST}_i$, a is the address of the G_{ij} in the memory zone of G_i such that $t \in \mathcal{TT}_{ij}$ and w is the index of the triangle corresponding to t in $A^{explicit}_{\kappa,\lambda}$ where $A_{\kappa,\lambda}$ is the triangulation to which G_{ij} points.

Similarly a vertex is represented by a triple (F_i, a, v). Observe however that this representation is not unique since, as opposed to triangles, verties may be shared by several tiny triangulations.

Operations on the triangulation Given a triangle (F_i, a, w) or a vertex (F_i, a, v) the operations $Triangle$, $Index$ and $Vertex$ can be implemented easily (just use the same operation in $A^{explicit}_{\kappa,\lambda}$).

The only difficulty is with $Neighbor((F_i, a, w), (F_i, a, v))$. If triangle w is not on the boundary of $A^{explicit}_{\kappa,\lambda}$, then just find its neighbor w' and return (F_i, a, w'). Otherwise, proceed as follows

— find in triangle w of $A^{explicit}_{\kappa,\lambda}$ the vertex v_{cw} following v in clockwise order in w.

— compute $l = Rank_1(v_{cw}) + 1$ in the bit vector associated to G_{ij}: it says that we are on the lth side of $\mathcal{TT}_{ij}$ ($l = 3$ in Figure 3);

— compute $l' = Select_1(l) - v_{cw}$: it says that v_{cw} is l'th before the end of the side ($l' = 1$ in Figure 3);

— let $x = G^{address}_{i,j,l}$, $y = G^{back}_{i,j,l}$ and $z = G^{small}_{i,j,l}$;

— if $z > 0$ let $G_{i'}$ be the submap of G pointed at by the zth neighbor $F_{i'}$ of F_i in F; otherwise let $G_{i'}$ be equal to G_i;

— let $G_{i',j'}$ be the node of G at address x in the memory zone of $G_{i'}$ and $A^{explicit}_{\kappa',\lambda'}$ the tiny triangulation it points at ($y = 5$ in Figure 3, the yth side of $G_{i',j'}$ matches the lth side of $G_{i,j}$);

— let $v'_{cw} = Select_1(y-1)+l'$ in the bit vector associated to $G_{i',j'}$: then v'_{cw} in $A^{explicit}_{\kappa',\lambda'}$ matches v_{cw} in $A^{explicit}_{\kappa,\lambda}$;

— let w' be the triangle pointed at by v'_{cw} in $A^{explicit}_{\kappa',\lambda'}$;

• return triangle $(F_{i'}, x, w')$.

8 Construction of the data structure

8.1 Exhaustive list of all tiny triangulations

Lemma 6. *The computation of Table A, and of all the information associated with Tables A_i can be done in $O(m^{0.81})$ time.*

Proof. Here we use the following result about optimal coding of planar triangulations [15]: a triangulation with n vertices can be coded by a bit string of length $4n-2$ and weight $n-1$.

The set of Tables A_i can be constructed by enumerating all the binary words of size $4i-2$ and weight $i-1$ in lexicographic order: this phase takes $O(\binom{4i}{3i} i)$ time, adopting a generating algorithm for combinations described in [11] (see 7.2.1.3, Vol. 4).

For each of these binary words, we try to decode it: if this phase fails the word is forgotten. Otherwise a triangulation with $2i$ faces is decoded: we now obtain a tiny triangulation $A_{k,j}$ with boundary by deleting the root vertex and its incident edges. If its size k is at most $\frac{1}{4}\lg m$ then $A_{k,j}$ is a valid tiny triangulation. Then we construct $A^{explicit}_{k,j}$ and we store a pointer to the explicit representation. Otherwise $\frac{1}{4}\lg m < k$ and we skip this word, since it yields a triangulation with too many faces.

Valid triangulations are sorted in increasing order of size to produce the tables A_i, each containing all triangulations of size i. These tables are in turn are sorted in lexicographic order of the corresponding binary word. The original binary words are stored in an auxiliary table A^{code}_i.[2]

Decoding a word requires $O(i) \leq O(\lg m)$ time, thus the cost for Table A_i is $O(\binom{4i-2}{i-1} \lg m) = O(2^{3.24i} \lg m)$ and after summing on i the whole construction cost is $O(m^{3.24\frac{1}{4}}) = O(m^{0.81})$.

Sorting valid triangulations according to their size takes in overall $O(\|A\| \lg \|A\|) = O(m^{\frac{1}{4}2.175} \lg m)$ time. □

[2] The length of this table is the same as A_i and the size of an element is $O(\lg n)$ as for A_i, thus Lemma 2 apply also to the storage needed for A^{code}_i.

8.2 Rank/Select on bit-vectors

Lemma 7. *The computation of Table B, and of all the information associated with Tables B_{pq} can be done in $O(m^{\frac{1}{4}} \lg m)$ time.*

Proof. As for Table A, this phase is based on the exhaustive enumeration of all bit-vectors of a given size and weight. Again let us fix $p, q \leq \frac{1}{4} \lg m$ and apply the previous algorithm for lexicographic generation of all binary words of size p and weight q. The construction of an explicit representation as described in Section 4 takes $O(\lg m)$ time: since there are at most $2^{\frac{1}{4} \lg m}$ bit-vectors, we need $O(m^{\frac{1}{4}} \lg m)$ time to compute these representations and store pointers to them in Table B_{pq}, the original bit vector is stored in an auxiliary table B_{pq}^{code}.[3] □

8.3 Maps of small and tiny triangulations

Lemma 8. *The computation of maps G and F can be done in $O(m)$ time.*

Proof. The triangulation $\mathcal{T}$ is decomposed in small and tiny triangulations in $O(m)$ time (Section 8.3.1), then explicit representations of G and F are constructed (Section 8.3.2). From that for each tiny triangulation, we compute the links to Tables A and B (Section 8.3.3). In a first traversal of the explicit version of G, we allocate the exact memory needed for the compact representation the nodes and in a second traversal the nodes are actually written in memory (Section 8.3.4). □

8.3.1 Decomposition into small and tiny triangulations

Let us suppose we are given a triangulation $\mathcal{T}$ consisting of less than m triangles. We can obtain its decomposition into small and tiny triangulations and the corresponding adjacency maps G and F in the following manner.

- Compute at first a spanning tree B of the dual $\mathcal{T}^*$: this is a binary tree B with m nodes (since each vertex in $\mathcal{T}^*$ has degree 3);
- apply to B the decomposition algorithm of Lemma 1 with parameter $M = \frac{1}{3} \lg^2 m$ obtaining a partition into small trees of size between $\frac{1}{3} \lg^2 m$ and $\lg^2 m$.
- For each small tree B_i, restart the partitioning with parameter $M = \frac{1}{12} \lg m$ obtaining a sub-partition into tiny trees of size between $\frac{1}{12} \lg m$ and $\frac{1}{4} \lg m$.

Tiny (resp. small) triangulations are obtained grouping triangles belonging to the same tiny (resp. small) tree: recall that each node in the tree corresponds to a triangle and each edge in the tree is the dual of an internal edge in the triangulation.

[3] The length of this table is the same as B_{pq} and the size of an element is $O(\lg n)$ as for B_{pq}, thus Lemma 3 apply also to the storage needed for B_{pq}^{code}.

Let us see how to construct boundaries of tiny (resp. small) triangulations and map G (resp. F): let us consider the set of triangles obtained as mentioned before.
• mark as *internal* all the edges which are dual of some edge in the tree;
• mark as *boundary* those edges that belong to at most one triangle in the tree (edges shared by triangles in two different subtrees);
• If there is an endpoint such that all incident edges are internal but one which is unclassified, then classify it as internal. Repeat this step as long as possible.

The remaining edges are classify as belonging to the boundary. In such a way, a tiny triangulation is always simply connected, although the boundary may be not simple. In such a case, the triangulation is incident to itself and there is a loop in G_i.

8.3.2 Construction of G and F

Once obtained a decomposition of the small triangulations into tiny triangulations let us construct the map G by associating to each tiny tree a node of G, and an arc in G for each side of the tiny triangulation.

The map G is simply the dual map of the map whose faces are limited by the computed boundary for tiny triangulations, and having one arc for each side of these boundaries. Multiple arcs are admitted, as well as loops, but faces have degree ≥ 3 because there is only one arc per side.

Construction of F is similar.

8.3.3 Computing the addresses in A and B

One crucial aspect of this phase concerns the representation of the combinatorial information of a tiny triangulation (stored in a node $G_{i,j}$) by its address in the exhaustive catalog of all tiny triangulations.

Let us consider a tiny triangulation t arising from the decomposition procedure: its size r, the size of its boundary p and the number q of its sides can be easily obtained (during the construction phase we have access to an explicit representation of the initial triangulation).

To associate to a node $G_{i,j}$ a representation of the corresponding tiny triangulation we proceed as follows.

Encode t using the same coder as in Section 8.1 [15]: the result is a binary word of length $4r-2$. This word is used to make a binary search in Table A_r^{code} and deduce the index of t in Table A_r which is the same as in Table A_r^{code}. This table contains about $2^{\frac{1}{4}2.175r}$ entries, and thus a binary search requires $\lg \|A_r^{code}\| = O(r)$ time.

Since in our case $r \leq \frac{1}{4}\lg m$ and there are about $\Theta(\frac{m}{\lg m})$ tiny triangulations, this phase takes $O(m)$ time.

A similar strategy can be applied for finding the implicit representation of boundary labelings in Table B. Once the boundary size p and the number q of sides are known, a binary search in Table B_{pq}^{code} is performed using the index of the relevant Rank/Select structure in Table B_{pq}. This takes $O(\lg m)$ time: again the overall computational cost of this phase is $O(m)$.

8.3.4 Computing the compact representation for G and F

The map F is a classical map represented using explicit pointers and its representation can be computed easily.

The representation for G is less easy. The compact representation is computed in turn for each submap G_i whose nodes are written consecutively in memory. For each node G_{ij} of G_i corresponding to a tiny triangulation $\mathcal{TT}_{ij}$ we use the right number of bits (as defined in Section 5.2) to write first the number of triangles in $\mathcal{TT}_{ij}$, the size of the boundary and the number of sides, then the index of the tiny triangulation and of the bit-vector encoding description of sides (as computed at the previous paragraph). Finally we allocate the memory to store the links to the neighbors of G_{ij} in G (the information to store a link to a neighbor has fixed size and the number of neighbors is known).

In a second step the map is traversed again, now all the nodes of the compact representation of map G have been allocated and the links between neighbors can be filled.

9 Entropy of planar triangulations with boundary

Our representation uses a table of all triangulations with at most $\frac{1}{4}\lg m$ (tiny triangulations). In order to bound the size of the pointers to this table we need to estimate the number of such triangulations.

More precisely we are interested in triangulations of a polygon with interior points such that there are no multiple edges but cords of the polygon are allowed. Let T_p^b be the set of these triangulations that are made of p triangles and $T_{k,p}^b$ be the subset of these that have a boundary polygon of size k. Observe that $p = 2n+k$ if $n+1$ denotes the number of internal vertices.

Mullin proved the following formula [12]:

$$|T_{k,2n+k}^b| = \frac{2\cdot(2k-3)!\,(2k+4n-1)!}{(k-1)!\,(k-3)!\,(n+1)!\,(2k+3n)!}, \tag{1}$$

In principle one should be able to derive a bound $|T_n^b|$ from this one by summation. It is however simpler to rederive the result in term of generating functions: define the formal power series $F(u,v)$ and $f_k(v)$ by

$$F(u,v) = \sum_{p,k} |T_{k,p}^b| u^k v^p \;=\; \sum_{k\geq 3} f_k(v)\, u^k.$$

Observe that the number $|T_p^b|$ are the coefficients in the Taylor expansion of $F(1,v)$ in the variable v. A decomposition of triangulations by root-edge deletion ([8, Sec 2.9.5]) yields the equation

$$F(u,v) = \frac{v}{u}(u^2 + F(u,v))^2 + \frac{v}{u}(F(u,v) - u^3 f_3(v)) - vF(u,v)f_3(v). \tag{2}$$

Equation 2 can be solved by the so-called quadratic method ([8, Sec 2.9.1]), resulting in the following expressions, which are equivalent to Mullin's formula: let $t \equiv t(v)$ be the unique solution analytic in zero of the equation

$$t(v) = 1 + v^2 t(v)^4.$$

Then $f_3(v) = v\,(2t^2 - t^3)$ and

$$F(1,v) = \frac{1}{2vt^2}(2 - 3t + 3vt^2 + \sqrt{1 + 7t - 9t^2 - 4vt^2 + 6vt^3 + 4vt^4}).$$

Since $F(1,v)$ is analytic at the origin and satisfy an algebraic equation, a standard approach to study its coefficients (*i.e.* the numbers of triangulations) is based on the analysis of its dominant singularity. On the one hand, the function $t(v)$ has a dominant singularity in $v = \rho = 3^{3/2}/2^4$, at which $t(\rho) = 4/3$. On the other hand, the square root in the expression of $F(1,v)$ yields another singularity for $v = \sigma = 2^6/17^2$, with $t(\sigma) = 17/16$. Since $\sigma < \rho$, σ is the dominant singularity. In the neighborhood of σ, $F(1,v)$ has singular expansion

$$F(1,v) = \frac{7}{8} - c\sqrt{1 - v/\sigma} + O((1 - v/\sigma)^{3/2}).$$

Using this expansion and Theorem [6, Thm VII.6] we obtain

$$|T_p^b| \;=\; c'\sigma^{-p}p^{-3/2}(1 + O(1/p)),$$

and taking the logarithm

$$\lg|T_p^b| \;=\; p\lg(1/\sigma) - \frac{3}{2}\lg p + \lg c' + O(1/p).$$

In conclusion there exists a constant c'' such that for all $p \geq 1$,

$$\lg|T_p^b| \;\leq\; p\,\lg(1/\sigma) + c'' \;\leq\; 2.175\,p + c''.$$

Combining numerical check for first terms and refined additivity arguments, one can check that the constant c'' can be in fact taken to be zero:

Lemma 9. *The number of triangulations with p faces is bounded by $2^{2.175p}$.*

Table 1 contains the number of triangulations for $p = 1..30$, and its lg compared to $2.175p$.

10 Concluding remarks and future work

10.1 Attaching information

The proposed structure represents only the connectivity of the triangulation. One may want to attach information to vertices or triangles, such as vertices coordinates (or colors...). This

p	$\lvert T_p^b\rvert$	$\lg \lvert T_p^b\rvert$	$2.175p$	p	$\lvert T_p^b\rvert$	$\lg \lvert T_p^b\rvert$	$2.175p$
1	1	1	2.175	16	222991801	29	34.800
2	2	2	4.350	17	924533609	31	36.975
3	6	4	6.525	18	3850615992	33	39.150
4	19	6	8.700	19	16104053458	35	41.325
5	66	8	10.875	20	67600318577	37	43.500
6	236	9	13.050	21	284729605627	40	45.675
7	877	11	15.225	22	1202959079012	42	47.850
8	3321	13	17.400	23	5096769502458	44	50.025
9	12840	15	19.575	24	21650252797852	46	52.200
10	50302	17	21.750	25	92186861004044	48	54.375
11	199657	19	23.925	26	393399791627096	50	56.550
12	800152	21	26.100	27	1682246653890199	52	58.725
13	3235923	23	28.275	28	7207306601612326	54	60.900
14	13182456	25	30.450	29	30933464929153561	56	63.075
15	54063790	27	32.625	30	132985945811160992	58	65.250

Table 1: First values

should be done by adding the information to nodes of G. For instance one can add to $G_{i,j}$ a table $G_{i,j}^{coordinate}$ describing the coordinates of the vertices of $\mathcal{TT}_{i,j}$. Table $G_{i,j}^{coordinate}$ contains the coordinates of all the internal vertices of $\mathcal{TT}_{i,j}$ and a selection of its boundary vertices, so that vertices lying on the side between two tiny triangulations are stored only once. It may happen that vertices belonging to more than two tiny triangulations have more than one copy: this does not really increase the space used, because it occurs at most $O(\frac{m}{\lg m})$ times (the argument is similar to the one used in the analysis of graph G). Basic compression on these coordinates can be obtained by giving them in a frame local to $G_{i,j}$.

10.2 Optimality versus class of triangulations

Our storage is optimal for the class of triangulations with boundary, which yields to the 2.175 bits per triangle. Other works often use the class of triangulations whose boundary is a triangle (triangulations of a 3d-sphere) which allows to link strongly the number of triangles and the number of vertices. In such triangulation, the optimum is 3.24 bits per vertex or equivalently 1.62 bits per triangle. Counting in number of bits per vertex in our structure is difficult since the vertices can be shared by several tiny triangulations.

10.3 Dynamic updates

In a forthcoming paper we will explain how this structure can be modified to allow for insertion of new vertices, flip and deletion of vertices of degree 3 in sub-linear time, in order to make dynamic updates of the triangulation possible.

10.4 Practical implementation

The result here is mainly theoretical: if m is one billion, $\frac{1}{4}\lg m$ is only 7. However the value $\frac{1}{4}$ is chosen to ensure that the table A can be constructed in sublinear time: looking at the actual number of triangulations with p faces for small p, one can check that constructing the table of all tiny triangulations up to size 13 is actually be feasible. In particular Table A can be computed once and for all and stored. We intend to implement and test a simplified version of this work, by gathering triangles in small groups of 3 to 5 triangles and making a map of these groups.

References

[1] D. Blanford, G. Blelloch, and I. Kash. Compact representations of separable graphs. In *Proc. of the Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 342–351, 2003.

[2] J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec. Triangulations in CGAL. *Comput. Geom. Theory Appl.*, 22:5–19, 2002.

[3] Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. Orderly spanning trees with applications to graph encoding and graph drawing. *SODA*, pages 506–515, 2001.

[4] R.C.-N Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. *Automata, Laguages and Programming*, pages 118–129, 1998.

[5] D. R. Clark and J. I. Raman. Efficient suffix trees on secondary storage. In *SODA*, pages 383–391, 2001.

[6] P. Flajolet and R. Sedgewick. Analytic combinatorics. Preliminary draft, 2004. `http://algo.inria.fr/flajolet/Publications/books.html`.

[7] R. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. In *SODA*, pages 1–10, 2004.

[8] Ian P. Goulden and David M. Jackson. *Combinatorial enumeration.* Dover Publications Inc., Mineola, NY, 2004. With a foreword by Gian-Carlo Rota, Reprint of the 1983 original.

[9] G. Jacobson. Space efficients static trees and graphs. In *Proceedings of the IEEE Symposium on Foundations of Computerb Science (FOCS)*, pages 549–554, 1989.

[10] M. Kallmann and D. Thalmann. Star-vertices: a compact representation for planar meshes with adjacency information. *Journal of Graphics Tools*, 6:7–18, 2002.

[11] D. Knuth. The art of computer programming. Preliminary draft, 2004. `http://www-cs-faculty.stanford.edu/~knuth/taocp.html`.

[12] R. C. Mullin. On counting rooted triangular maps. *Canad. J. Math*, 17:373–382, 1965.

[13] J. I. Munro and V. Raman. Succint representation of balanced parantheses and static trees. *SIAM J. on Computing*, 31:762–776, 2001.

[14] J. I. Munro, V. Raman, and A. J. Storm. Representing dynamic trees succintly. In *SODA*, pages 529–536, 2001.

[15] D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. In *Proc. Intern. Colloqium ICALP'03*, pages 1080–1094, 2003.

[16] R. Raman, V. Raman, and S.S. Rao. Succint indexable dictionaries with application to encoding k-ary trees and multisets. In *SODA*, pages 233–242, 2002.

[17] R. Raman and S.S. Rao. Static dictionaries supporting rank. In *ISAAC*, pages 18–26, 1999.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399