Dynamic updates of succinct triangulations*

Luca Castelli Aleardi[†] Olivier Devillers[‡] Gilles Schaeffer[§]

Abstract

In a recent article, we presented a succinct representation of triangulations that supports efficient navigation operations. Here this representation is improved to allow for efficient local updates of the triangulations.

Precisely, we show how a succinct representation of a triangulation with m triangles can be maintained under vertex insertions in $O(1)$ amortized time and under vertex deletions/edge flips in $O(\lg^2 m)$ amortized time.

Our structure achieves the information theory bound for the storage for the class of triangulations with a boundary, requiring asymptotically $2.17m + o(m)$ bits, and supports adjacency queries between triangles in $O(1)$ time (an extra amount of $O(g \lg m)$ bits are needed for representing triangulations of genus g surfaces).

1 Introduction

Data structures are usually based on explicit pointer representations. For instance, a binary tree is typically implemented using for each node a pointer to its left and right sons: $O(m)$ pointers of size $O(\lg m)$ bits are used to represent a tree with m nodes. Since there are less than 4^m different such trees, $2m$ bits should be enough. This observation leads, given a class $\mathcal{C}_m$ of objects of size m, to the question whether these objects can have a *succinct dynamic representation*. The aim is to design an updatable data structure R that allows efficient information retrieval and whose storage matches asymptotically the entropy: $\text{size}(R) = \log_2 |\mathcal{C}_m| \cdot (1 + o(1))$.

While there is a large literature on compact data structures, dynamic succinct data structures were only developed recently, for some basic fundamental data structures like dynamic arrays [5], [10], dynamic dictionaries [11], dynamic bit-vectors [10]. Our interest is in geometric data structures: here we consider the problem of representing succinctly the connectivity of triangulations of a surface of genus g with a simple boundary. Our main result is the following:

Theorem 1 *There exists a representation of triangulations of a surface of fixed genus g with a simple boundary using asymptotically at first order 2.17 bits per triangles, thus matching the entropy of this class of triangulations. For a triangulation with m triangles our representation requires $2.17m + O(g \lg m) + o(m)$ bits, for the storage. Navigation is supported in $O(1)$ time, while vertex addition requires $O(1)$ amortized time (amortized $O(\lg m)$ time with data access); $O(\lg^2 m)$ amortized time is needed for vertex deletion or edge flip.*

This *dynamic* result extends a previous *static* result [2] in which update operations were not dealt with. To our knowledge, this is the first dynamical version of succinct data structures for triangulations or complex graphs structures: related previous results are static or non succinct, or deal with simpler structures like trees. For dynamic binary trees with n nodes, an optimal succinct representation has been proposed [11, 9] allowing basic modifications on the nodes, while navigation operations are performed in $O(1)$ time. If external $O(\lg n)$ bits data are associated to nodes, adding/deleting a leaf or inserting a node along an edge require $O(\lg^2 n)$ amortized time in [9]. The cost of update was later improved to $O((\lg \lg n)^{1+\varepsilon})$ [11]. Compact representations for static planar graphs were designed [7, 8, 6], combining succinct representations for trees with combinatorial decompositions of planar graphs as four pages embedding or canonical orderings. This yields space-efficient solutions that allow to perform local navigation in $O(1)$ time, but the use of non trivial combinatorial decompositions makes it hard to maintain the structure under local modifications of the graph. With a different approach, based on short separators, a compact representation for separable graphs was developed in [3] that uses $O(n)$ bit, supports local navigation in $O(1)$ time and for which updates appear possible in practice: however it is difficult to give good bounds on the cost of updates in this approach, again because the separators are hard to track under local modifications.

Our approach is similar to the hierarchical 3 level approach used in [9, 10] for binary trees: we decompose the structure into *small* regions themselves cut into *tiny* pieces. However, these works can use canonical total orderings like the standard preorder on trees, which is relatively stable under insertion/deletion of leaves. This fails on graphs, because canonical orders on vertices are typically perturbed at large distance by local modifications of the graph: as illustrated by Fig. 1, this is the main new difficulty we have to deal with.

*This work has been supported by the French "ACI Masses de données" program, via the Geocomp project, http://www.lix.polytechnique.fr/~schaeffe/GeoComp/

[†]INRIA & LIX, amturing@lix.polytechnique.fr

[‡]INRIA, olivier.devillers@sophia.inria.fr

[§]LIX, École Polytechnique, schaeffe@lix.polytechnique.fr

Contribution Our main contribution, as stated in Theorem 1 above, is to prove that we can theoretically update and traverse a triangulation with good complexity using an asymptotically optimal storage. We also discuss the cost of providing access to attached data (like vertex coordinates).

Practical implementations [4] are far from that and use 6 pointers (i.e. $6 \cdot 32 = 192$ or $6 \log m$ bits) per triangle. Even if this work is clearly theoretical, it suggests ideas for more practical trade-offs between storage and access time.

Overview of the solution As in previous work on succinct representations for static and dynamic binary trees, our structure relies on a 3 level description of the initial triangulation $\mathcal{T}$.

The underlying level consists of a set of sub-triangulations of size $\Theta(\lg m)$ that is a partition of the m triangles of $\mathcal{T}$: we will call *tiny triangulations*, denoted by $\mathcal{T}_{ij}$ (and stored in Table A), those sub-triangulations having between $\frac{1}{12}\lg m$ and $\frac{1}{4}\lg m$ triangles. Their tiny size ensures that we can store the catalog of all possible different triangulations having up to $\frac{1}{4}\lg m$ triangles using a sub-linear amount of space. Each tiny triangulation is a planar triangulation with a boundary of arbitrary size (no assumptions are made on the way we partition the m triangles of $\mathcal{T}$). We call *multiple vertices* those boundary vertices that are shared by more than 2 tiny triangulations; these multiple vertices cut the boundary of tiny triangulations in pieces called *sides*.

Tiny triangulations are packed together to form bigger connected triangulations of about $\Theta(\lg^2 m)$ triangles, called *small triangulation* and denoted $\mathcal{ST}_i$. A small triangulation contains between $\frac{1}{3}\lg m$ and $\lg m$ tiny triangulations.

The second level of the structure is designed to describe adjacency relations between tiny triangulations: this is done by introducing a graph G whose nodes and arcs correspond to tiny triangulations and neighborhood relationships between them. G is a planar map, whose faces have degree at least 3 and having loops and multiple edges (auto-intersections of boundaries are allowed). A small triangulation $\mathcal{ST}_i$ maps to a group of size $\Theta(\lg m)$ of nodes of G, denoted by G_i.

The upper level is needed to describe adjacency relations between small triangulations: for this purpose we introduce a graph F, represented with true pointers on $O(\lg m)$ bits, which links adjacent small triangulations.

As described in [1, 2], F is represented explicitly using $\lg m$ size pointers: this requires sub-linear storage since $|F| = O\left(m/\lg^2 m\right)$. As opposed to that, adjacencies in G are represented by pointers of size $\lg \lg m$ that are local to each small region G_i (which has size $|G_i| = O(\lg m)$). The overall cost of these local pointers also sum up to a sub-linear storage. Finally the dominant linear cost arises from the storage for each vertex of G of a pointer to a tiny triangulation in Table A.

The overall succinctness ultimately comes from the fact that a tiny triangulation appearing several times in $\mathcal{T}$ is explicitly stored only once, with several nodes of G pointing at it.

2 Preliminaries and previous results used

As model of computation we consider a standard RAM machine, where memory is dynamically allocated and that supports access and arithmetic operations in $O(1)$ time on words of size $\lg m$. Our previous representation [2] supported access from a triangle to its neighbors in $O(1)$ time, the extension in this paper allows to perform local modifications of the triangulation:
- $Insert(\triangle)$: adds a degree 3 vertex in triangle $\triangle$.
- $Delete(v)$: deletes a degree 3 vertex.
- $Flip(\triangle, v)$: flips the edge of $\triangle$ opposite to vertex v.

Succinct dynamic arrays A key tool for managing memory is a dynamic data structure called *extensible array*. An extensible array contains n equal-size records (with index between 0 and $n-1$) and supports static and dynamic operations: accessing a record given its index, creating/discarding a new record with index n (*grow/shrink*). To manage records of variable sizes, we can use a collection of a arrays having relevant size records r_i between a maximum and a minimum of nominal size $s = \sum_{i=1}^{a} n_i r_i$. Our memory storage is based on some recent result presented in [11] and expressed by:

Lemma 2 *Let us consider a collection of a extensible arrays whose nominal size is s, and denote by w the machine word size. Then there exists a succinct representation of the collection that uses $s + O(aw + \sqrt{saw})$ bits of space and supports* access *operations in $O(1)$ time and dynamic operations in $O(1)$ amortized time.*

Succinct triangulations The following aspects of the static version [2] remain unchanged here.
- The **number of triangulations** of a polygon, with p triangles using interior vertices but not multiple edges is bounded by $2^{2.175p}$.
- The **catalog of all tiny triangulations** having $k(<\frac{1}{4}\lg m)$ triangles is stored in Table A_k containing pointers to an explicit pointer based representation of the triangulation. In this paper, the representation of each tiny triangulation $\mathcal{TT}$ is augmented by links to the other tiny triangulations obtained from $\mathcal{TT}$ by *Insert*, *Delete* or *Flip* operations.
- Together with the pointer into Table A and the list of its neighbors, each node of G has a pointer into a **catalog of boundary bit vectors**, that allows to encode

how the boundary of the tiny triangulation is split into sides. Table B_{pq} contains all bit-vectors of size p and weight q (for $p < \frac{1}{4}\lg m$) that can occur at a degree q node whose tiny triangulation has a boundary of size p.

Tables A_i (resp. B_{pq}) contain $O(2^{2.17\frac{1}{4}\lg m})$ (resp. $O(m^{\frac{1}{4}})$) entries of polylogarithmic size: their storage thus requires a negligible amount of space.

3 Map of tiny triangulations

The map G describes the adjacency relations between tiny triangulations, locally within each submap G_i corresponding to nodes in a same small triangulation $\mathcal{ST}_i$, and that the map F has the G_i as vertices and describes their adjacency relations. We allow the maps G and F to have multiple arcs or loops but all their 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 $\mathcal{TT}_{j'}$.

We now give a detailed description of 5 collections of extensible arrays associated with a node G_i. The first one stores the neighbors of G_i in F, while the other four describe the submap G_i:

• An extensible array S_i is used to store adjacency relations between small triangulations (represented by arcs in map F): this array has $O(\lg^2 m)$ records, each of size $\Theta(\lg m)$ bits, listing the neighbors of the sub-map G_i.

• An extensible array T_i is used to store information relative to each tiny triangulation: the record $T_i[j]$, relative to the tiny triangulation $\mathcal{TT}_{i,j}$ consists of the $O(\lg\lg m)$ bit concatenation of the following fields:

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

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

– $G^s_{i,j}$ is the degree of the node $G_{i,j}$ in the map G_i.

– $G^{PA}_{i,j}$ is a reference to the record, in an array of the collection PA_i below, where the combinatorial structure of $\mathcal{TT}_{ij}$ is implicitly stored as a reference[1] in Table $A_{G^t_{i,j}}$.

– $G^{PB}_{i,j}$ is a reference to the record, in an array of the collection PB_i below, where the boundary labeling of $\mathcal{TT}_{ij}$ is implicitly stored as a reference in Table $B_{G^b_{i,j},G^s_{i,j}}$.

– $G^E_{i,j}$ is a reference, in an array of the collection PE_i below, to the list of halfarcs incident to node G_{ij}.

Each of these fields can be represented on $O(\lg\lg m)$ bits, since the submap G_i has at most $O(\lg m)$ nodes and $O(\lg^2 m)$ arcs [2].

• A collection PA_i of extensible arrays is used to store implicitly (as references to Table A) the combinatorial structures of the tiny triangulations of G_i. The collection PA_i consists of $O(\sqrt{\lg m})$ extensible arrays: the kth array in this collection has records of size $k\sqrt{\lg m}$. Each record is meant to contain a pair of the form:

[1] This reference cannot be stored directly since it uses a non constant number of bits, namely $2.175 G^t_{i,j}$ bits.

– The index of a tiny triangulation in Table A. This index has size $2.175r$ bits for a triangulation of size r.

– A $\lg\lg n$ bits index used as backward pointer in table T_i.

Data pairs will be added, removed and modified in PA_i but we shall ensure that each pair is always assigned to the array with smallest index k among arrays with large enough records for it to fit.

• A collection PB_i of extensible arrays is used to store implicitly (as references to Table B) the boundary bit vectors. The collection PB_i is modeled after PA_i with the only difference that it contains indices of boundary bit vectors in Table B, instead of indices of tiny triangulations. For a tiny triangulation with boundary size p and q sides, these references require $O(q\lg p)$ bits.

• A collection PE_i of extensible arrays is used to store the (ccw sorted) lists of half arcs incident to the nodes $G_{i,j}$. For each half-arc, the following $O(\lg\lg m)$ bit fields are stored:

– $G^T_{ij}.source$ is a reference in Table T_i to the record associated with G_{ij} (source node of the arc);

– $G^T_{ij}.target$ is a reference in Table T_i to the record associated with $G_{i'j'}$, which is pointed by the halfarc ($G_{i'j'}$ is the target of the halfarc);

– $G_{ij}.back$ is the index k' of the side corresponding to the current arc in the numbering of sides at the opposite node $G_{i'j'}$.

– $G_{ij}.small$ is the index of the small triangulation $G_{i'}$, in the extensible array S_i of the neighbors of G_i in map F.

A node $G_{i,j}$ can have degree between 1 and $\lg m$, so that it requires at most $O(\lg m \lg\lg m)$ bits.

The collection PE_i is thus taken of $\ell = \sqrt{\lg m \lg\lg m}$ arrays with records of size $k\ell$ for $k = 1, \ldots, \ell$. Each node is stored in a record of the array with smallest index having large enough records. In this way, $O(\ell)$ bits per edge are wasted, but this does not matter since the total number of edges in G is $O(m/\lg m)$.

Lemma 3 *The storage of maps G and F requires asymptotically $2.175m + o(m)$ bits*

For a detailed proof see [1].

4 Update of the triangulation

In this section we describe 2 subroutines used in the updating of the triangulation: they are called when the size of a tiny triangulation goes outside the prescribed bounds, for example after a vertex insertion.

Splitting a tiny triangulation The first procedure is designed to perform the decomposition and the update of a non valid tiny triangulation whose size exceeds the upper bound of $\frac{1}{4}\lg m$ triangles. By cutting at the right

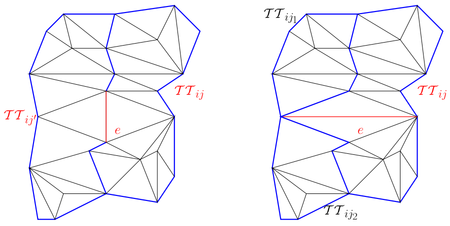

Figure 1: A region disconnecting flip

place a spanning tree of the dual of the triangulation, we prove (see [1]):

Lemma 4 *Splitting a triangulation having between $\frac{1}{4}\lg m$ and $\frac{3}{8}\lg m$ triangles into two valid tiny triangulations requires $O(\lg m)$ amortized time.*

Splitting a small triangulation When the number of tiny triangulations in a small triangulation exceeds the allowed maximum, a procedure for decomposing the small triangulation into two parts is required.

A naive splitting leads to expensive updates for the potentially numerous neighbors of the small triangulation. However we claim that, using a more clever decomposition and update scheme, only one tiny triangulation needs to be split, so that a negligible number of adjacency relations in map G have to be updated.

Lemma 5 *Splitting a small triangulation $\mathcal{ST}_i$ together with updating all data structures involved requires $O(\lg^2 m)$ amortized time.*

The proofs of lemmas above are detailed in [1]. The reverse operations, **Join** of two tiny or of two small triangulations, can be done at the same cost.

Update of the triangulation The three update operations that are supported are the insertion or the deletion of a vertex or the flipping an edge. When such an operation occurs inside a tiny triangulation (that is, without changing its boundary), the modification is done in constant time since the new tiny triangulation is precomputed in Table A, and only a reference in PA_i changes. If the operation leads to a violation of the prescribed bounds on tiny triangulation sizes, Lemma 4 is used.

When only insertions are used such splittings can occur only every $\Theta(\lg m)$ insertions, so that the cost can be amortized. In the same way, if the splitting of the tiny triangulation increases the number of tiny triangulations inside the small triangulation, Lemma 5 is used, and its cost can be amortized over successive insertions.

Vertex deletions and splitting can be more intricate because they may involve several tiny or small triangulations when the deleted vertex or the flipped edge is on the boundary. In such a case, the involved triangulations are first unified using the **Join** operation, the deletion or the flip is performed and the resulting triangulation is then **Split** so that the sizes obey the prescribed bounds.

Even if these join and split operations should occur rarely in practice, it is no possible to amortized their cost: indeed vertex deletions or edge flips may impose the splittings of tiny or small triangulations arbitrarily often for topological reasons (see for instance Fig. 1), as opposed to splitting due to size bound violation which cannot occur frequently. To summarize, we get:

Lemma 6 *Performing an edge flip or a vertex deletion in $\mathcal{T}$ requires $O(\lg^2 m)$ amortized time. Vertex insertions take constant amortized time.*

5 Attaching geometric information

A purely combinatorial structure is of little interest, we often need to attach information to triangles or vertices: for example the vertices coordinates. This can be done by attaching to G_i a new extensible array which has to be updated when the triangulation is modified. For the bare combinatorial structure, an insertion that occurs inside a tiny triangulation requires only a constant time pointer substitution, but this modification of a tiny triangulation induces a local renumbering of its vertices which in turn may require to reorganize the attach information at an extra $O(\lg m)$ time cost.

Lemma 7 *If external data on $O(\lg m)$ bits are associated to vertices of $\mathcal{T}$, the structure can be maintained under vertex insertions in $O(\lg m)$ amortized time. Edge flips and vertex deletions can be performed in $O(\lg^2 m)$ amortized time.*

References

[1] L. Castelli Aleardi, O. Devillers, and G. Schaeffer. Dynamic updates of succinct triangulations. INRIA report, 2005.[a]

[2] L. Castelli Aleardi, O. Devillers, and G. Schaeffer. Succinct representation of triangulations with a boundary. *WADS*, 2005.[a]

[3] D. Blanford, G. Blelloch, and I. Kash. Compact representations of separable graphs. *SODA* 342–351, 2003.

[4] J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec. Triangulations in CGAL. *CGTA*, 22:5–19, 2002.

[5] A. Brodnik, S. Carlsson, E. Demaine, J. Munro, and R. Sedgewick. Resizable arrays in optimal time and space. *WADS*, 37–48, 1999.

[6] R. Chuang, A. Garg, X. He, M. Kao, and H. Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. *Automata, Languages and Programming*, 118–129, 1998.

[7] G. Jacobson. Space efficients static trees and graphs. *FOCS*, 549–554, 1989.

[8] J. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM JoC*, 31:762–776, 2001.

[9] J. Munro, V. Raman, and A. Storm. Representing dynamic binary trees succinctly. *SODA*, 529–536, 2001.

[10] R. Raman, V. Raman, and S. Rao. Succinct dynamic data structures. *WADS*, 426–437, 2001.

[11] V. Raman and S. Rao. Succinct dynamic dictionaries and trees. *ICALP*, 357–366, 2003.

[a] `http://www.lix.polytechnique.fr/~amturing/publications.html`