The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes

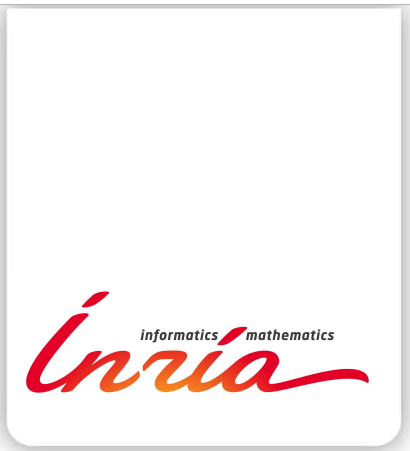
Jean-Daniel Boissonnat, Clément Maria

The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes

Jean-Daniel Boissonnat, Clément Maria

RESEARCH REPORT
N° 7993
June 2012
Project-Team GEOMETRICA

ISSN 0249-6399 ISRN INRIA/RR--7993--FR+ENG



The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes

Jean-Daniel Boissonnat*, Clément Maria†

Project-Team GEOMETRICA

Research Report n° 7993 — June 2012 — 20 pages

Abstract: This paper introduces a new data structure, called simplex tree, to represent abstract simplicial complexes of any dimension. All faces of the simplicial complex are explicitly stored in a trie whose nodes are in bijection with the faces of the complex. This data structure allows to efficiently implement a large range of basic operations on simplicial complexes. We provide theoretical complexity analysis as well as detailed experimental results. We more specifically study Rips and witness complexes.

Key-words: simplicial complexes, data structure, flag complexes, Rips complexes, witness complexes, relaxed witness complexes, high dimensions

* INRIA Sophia Antipolis-Méditerranée
† ENS Cachan

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Une structure de données arborescente pour représenter les complexes simpliciaux

Résumé : Nous définissons dans cet article une nouvelle structure de données, appelée "simplex tree", pour représenter les complexes simpliciaux abstraits de toutes dimensions. Le complexe simplicial est représenté par un arbre préfixe dont les nœuds sont en bijection avec les faces du complexe. Cette structure de données permet de calculer efficacement un grand nombre d'opérations de bases sur les complexes simpliciaux. Nous développons dans cet article une analyse théorique de la complexité de ces algorithmes, ainsi qu'une analyse expérimentale détaillée. Nous étudions plus particulièrement la construction des complexes de Rips et des witness complexes.

Mots-clés : complexes simpliciaux, structure de données, flag complexes, complexes de Rips, witness complexes, relaxed witness complexes, grandes dimensions

1 Introduction

Simplicial complexes are widely used in combinatorial and computational topology, and have found many applications in topological data analysis and geometric inference. A variety of simplicial complexes have been defined, for example the Čech complex, the Rips complex and the witness complex [9, 8]. However, the size of these structures grows very rapidly with the dimension of the data set, and their use in real applications has been quite limited so far.

We are aware of only a few works on the design of data structures for general simplicial complexes. Brisson [5] and Lienhardt [13] have introduced data structures to represent d-dimensional cell complexes, most notably subdivided manifolds. While those data structures have nice algebraic properties, they are very redundant and do not scale to large data sets or high dimensions. Zomorodian [18] has proposed the tidy set, a compact data structure to simplify a simplicial complex and compute its homology. Since the construction of the tidy set requires to compute the maximal faces of the simplicial complex, the method is especially designed for flag complexes. Flag complexes are a special type of simplicial complexes (to be defined later) whose combinatorial structure can be deduced from its graph. In particular, maximal faces of a flag complex can be computed without constructing explicitly the whole complex. In the same spirit, Attali et al. [2] have proposed the skeleton-blockers data structure. Again, the representation is general but it requires to compute blockers, the simplices which are not contained in the simplicial complex but whose proper subfaces are. Computing the blockers is difficult in general and details on the construction are given only for flag complexes, for which blockers can be easily obtained. As of now, there is no data structure for general simplicial complexes that scales to dimension and size. The best implementations have been restricted to flag complexes.

Our approach aims at combining both generality and scalability. We propose a tree representation for simplicial complexes. The nodes of the tree are in bijection with the simplices (of all dimensions) of the simplicial complex. In this way, our data structure explicitly stores all the simplices of the complex but does not represent explicitly all the adjacency relations between the simplices. Storing all the simplices provides generality, and the tree structure of our representation enables us to implement basic operations on simplicial complexes efficiently, in particular to retrieve incidence relations. Moreover, storing exactly one node per simplex ensures that the size of the structure adapts to the intrinsic complexity of the simplicial complex to be represented.

1.1 Background

Simplicial complexes. A *simplicial complex* is a pair $\mathcal{K} = (V, S)$ where V is a finite set whose elements are called the *vertices* of $\mathcal{K}$ and S is a set of non-empty subsets of V that is required to satisfy the following two conditions :

1. $p \in V \Rightarrow \{p\} \in S$
2. $\sigma \in S, \tau \subset \sigma \Rightarrow \tau \in S$

Each element $\sigma \in S$ is called a *simplex* or a *face* of $\mathcal{K}$ and, if $\sigma \in S$ has precisely $s+1$ elements ($s \geq -1$), σ is called an s-simplex and the dimension of σ is s. The dimension of the simplicial complex $\mathcal{K}$ is the largest k such that S contains a k-simplex.

We define the j-skeleton, $j \geq 0$, of a simplicial complex $\mathcal{K}$ to be the simplicial complex made of the faces of $\mathcal{K}$ of dimension at most j. In particular, the 1-skeleton of $\mathcal{K}$ contains the vertices

and the edges of $\mathcal{K}$. The 1-skeleton has the structure of a graph, and we will equivalently talk about the graph of the simplicial complex, and use graph notations to describe its combinatorial structure.

Faces and cofaces. A *face* of a simplex $\sigma = \{p_0, \cdots, p_s\}$ is a simplex whose vertices form a subset of $\{p_0, \cdots, p_s\}$. A *proper* face is a face different from σ and the *facets* of σ are its proper faces of maximal dimension. A simplex $\tau \in \mathcal{K}$ admitting σ as a face is called a *coface* of σ. The subset of simplices consisting of all the cofaces of a simplex $\sigma \in \mathcal{K}$ is called the *star* of σ.

The *link* of a simplex σ in a simplicial complex $\mathcal{K}$ is defined as the simplicial complex $\{\tau \in \mathcal{K} | \sigma \cup \tau \in \mathcal{K}, \sigma \cap \tau = \emptyset\}$.

Filtration A *filtration* over a simplicial complex $\mathcal{K}$ is an ordering of the simplices of $\mathcal{K}$ such that all prefixes in the ordering are subcomplexes of $\mathcal{K}$. In particular, for two simplices τ and σ in the simplicial complex such that $\tau \subsetneq \sigma$, τ appears before σ in the ordering. Such an ordering may be given by a real number associated to the simplices of $\mathcal{K}$. The order of the simplices is simply the order of the real numbers.

2 Simplex Tree

In this section, we introduce a new data structure which can represent any simplicial complex. This data structure is a trie [4] which will explicitly represent all the simplices and will allow efficient implementation of basic operations on simplicial complexes.

2.1 Simplicial Complexes and Trie

Let $\mathcal{K}$ be a simplicial complex of dimension k, V its vertex set. The vertices are labeled from 1 to $|V|$ and ordered accordingly.

We can thus associate to each simplex of $\mathcal{K}$ a word on the alphabet $1 \cdots |V|$. Specifically, a j-simplex of $\mathcal{K}$ is uniquely represented as the word of length $j+1$ consisting of the ordered set of the labels of its $j+1$ vertices. Formally, let simplex $\sigma = \{v_{\ell_0}, \cdots, v_{\ell_j}\}$, where $v_{\ell_i} \in V$, $\ell_i \in \{1, \cdots, |V|\}$ and $\ell_0 < \cdots < \ell_j$. σ is represented by the word $[\sigma] = [\ell_0, \cdots, \ell_j]$. The last label of the word representation of a simplex σ will be called the last label of σ and denoted by $last(\sigma)$.

The simplicial complex $\mathcal{K}$ can be defined as a collection of words on an alphabet of size $|V|$. To compactly represent the set of simplices of $\mathcal{K}$, we store the corresponding words in a tree satisfying the following properties:

1. The nodes of the simplex tree are in bijection with the simplices (of all dimensions) of the complex. The root is associated to the empty face.

2. Each node of the tree, except the root, stores the label of a vertex. Specifically, a node N associated to simplex $\sigma \neq \emptyset$ stores the label of vertex $last(\sigma)$.

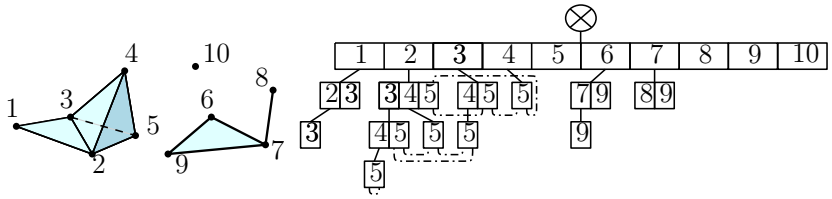


Figure 1: A simplicial complex on 10 vertices and its simplex tree. The deepest node represents the tetrahedron of the complex. All the positions of a given label at a given depth are linked in a list, as illustrated in the case of label 5.

3. The vertices whose labels are encountered along a path from the root to a node N, associated to a simplex σ, are the vertices of σ. The labels are sorted by increasing order along such a path, and each label appears exactly once.

We call this data structure the *Simplex Tree* of $\mathcal{K}$. It may be seen as a trie [4] on the words representing the simplices of the complex (Figure 1). The depth of the root is 0 and the depth of a node is equal to the dimension of the simplex it represents plus one.

In addition, we augment the data structure so as to quickly locate all the instances of a given label in the tree. Specifically, all the nodes at a same depth j which contain a same label ℓ are linked in a circular list $L_j(\ell)$, as illustrated in Figure 1 for label $\ell = 5$.

We also attach to each set of sibling nodes a pointer to their parent so that we can access a parent in constant time.

The children of the root of the simplex tree are called the *top nodes*. The top nodes are in bijection with the elements of V, the vertices of $\mathcal{K}$. Nodes which share the same parent (e.g. the top nodes) will be called *sibling nodes*.

We give a constructive definition of the simplex tree. Starting from an empty tree, we insert the words representing the simplices of the complex in the following manner. When inserting the word $[\sigma] = [\ell_0, \cdots, \ell_j]$ we start from the root, and follow the path containing successively all labels $\ell_0, \cdots, \ell_i$, where $[\ell_0, \cdots, \ell_i]$ denotes the longest prefix of $[\sigma]$ already stored in the simplex tree. We then append to the node representing $[\ell_0, \cdots, \ell_i]$ a path consisting of the nodes storing labels $\ell_{i+1}, \cdots, \ell_j$.

It is easy to see that the three properties above are satisfied. Hence, if $\mathcal{K}$ consists of $|\mathcal{K}|$ simplices (including the empty face), the associated simplex tree contains exactly $|\mathcal{K}|$ nodes.

We use dictionaries for searching, inserting and removing elements among a set of sibling nodes. As the size of a dictionary is linear in the number of elements it stores, these additional structures do not change the asymptotic complexity of the simplex tree. For the top nodes, we simply use an array since the set of vertices V is known and fixed. Let $d^o_{\max}$ denote the maximal outdegree of a node in the tree distinct from the root. Remark that $d^o_{\max}$ is at most the maximal degree of a vertex in the graph of the simplicial complex. In the following, we will denote by D_m the maximal number of operations needed to perform a search, an insertion or a removal in a dictionary of maximal size $d^o_{\max}$ (for example, with red-black trees $D_m = O(\log(d^o_{\max}))$). Some algorithms, that we describe later, on the simplex tree require to intersect and to merge sets of sibling nodes. In this case, we will prefer dictionaries keeping their elements sorted to allow fast

set intersections (e.g., red-black trees).

The degree of a vertex in the simplicial complex depends on the inherent structure of the data. Let us define the doubling dimension of a set S in $\mathbb{R}^D$ to be the minimal integer d such that for any $x \in \mathbb{R}^D$ and any $r > 0$ we can cover the set $B(x,r) \cap S$ with 2^d balls of radius $\frac{r}{2}$, where $B(x,r)$ denotes the closed ball centered at x and of radius r. The doubling dimension captures the intrinsic dimensionality of the data. The following illustration is meaningful in the context of geometric and topological data analysis. Let $\mathbb{M}$ be a d-manifold with bounded curvature, embedded in $\mathbb{R}^D$. $\mathbb{M}$ admits a $O(d)$ doubling dimension [7], and so does any set of points sampled from the manifold. Consequently, for a set of points S sampled from $\mathbb{M}$, the degree of the vertices of a geometric simplicial complex, with vertex set S and homeomorphic to $\mathbb{M}$, is $O(2^d)$. Hence, the time to perform an elementary operation in a dictionary depends on the intrinsic dimension of the set of vertices (for example, $D_m = O(d)$ for red-black trees).

We introduce two new notations for the analysis of the complexity of the algorithms. Given a simplex $\sigma \in \mathcal{K}$, we define C_σ to be the number of cofaces of σ. Note that C_σ only depends on the combinatorial structure of the simplicial complex $\mathcal{K}$. Let $\mathcal{T}$ be the simplex tree associated to $\mathcal{K}$. Given a label ℓ and an index j, we define $\mathcal{T}_\ell^{>j}$ to be the number of nodes of $\mathcal{T}$ at depth strictly greater than j that store label ℓ. These nodes represent the simplices of dimension greater than j that admit ℓ as their last label. $\mathcal{T}_\ell^{>j}$ depends on the labelling of the vertices and is bounded by $C_{\{v_\ell\}}$, the number of cofaces of the vertex with label ℓ. For example, if ℓ is the greatest label, we have $\mathcal{T}_\ell^{>0} = C_{\{v_\ell\}}$, and if ℓ is the smallest label we have $\mathcal{T}_\ell^{>0} = 1$ independently from the number of cofaces of $\{v_\ell\}$.

2.2 Operations on a Simplex Tree

2.2.1 Insertions and Adjacency Retrieval

Insertions and Removals Using the previous top-down traversal, we can *search* and *insert* a word of length j in $O(jD_m)$ operations.

We can extend this algorithm so as to insert a simplex and all its subfaces in the simplex tree. Let σ be a simplex we want to insert with all its subfaces. Let $[\ell_0, \cdots, \ell_j]$ be its word representation. For i from 0 to j we insert, if not already present, a node N_{ℓ_i}, storing label ℓ_i, as a child of the root. We recursively call the algorithm on the subtree rooted at N_{ℓ_i} for the insertion of the suffix $[\ell_{i+1}, \cdots, \ell_j]$. Since the number of subfaces of a simplex of dimension j is $\sum_{i=0\cdots j+1} \binom{j+1}{i} = 2^{j+1}$, this algorithm takes time $O(2^j D_m)$.

We can also remove a simplex from the simplex tree. Note that to keep the property of being a simplicial complex, we need to remove all its cofaces as well. We locate them thanks to the algorithm described below.

Locate cofaces. Computing the cofaces of a face is required to retrieve adjacency relations between faces. In particular, it is useful when traversing the complex or when removing a face. We also need to compute the cofaces of a face when contracting an edge (described later) or during the construction of the witness complex, described later in section 3.2.

If τ is represented by the word $[\ell_0 \cdots \ell_j]$, the cofaces of τ are the simplices of $\mathcal{K}$ which are represented by words of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j\star]$, where $\star$ represents an arbitrary word on the alphabet, possibly empty.

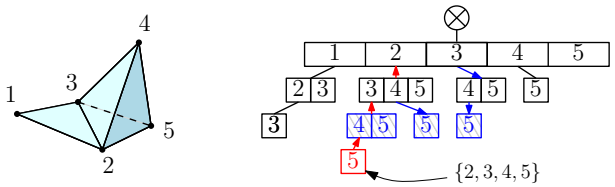


Figure 2: Facets location of the simplex $\sigma = \{2, 3, 4, 5\}$, starting from the position of the σ in the simplex tree. The nodes representing the facets are colored in grey.

To locate all the words of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j\star]$ in the simplex tree, we first find all the words of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j]$. Using the lists $L_i(\ell_j)$ $(i > j)$, we find all the nodes at depth at least $j+1$ which contain label ℓ_j. For each such node N_{ℓ_j}, we traverse the tree upwards from N_{ℓ_j}, looking for a word of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j]$. If the search succeeds, the simplex represented by N_{ℓ_j} in the simplex tree is a coface of τ, as well as all the simplices represented by the nodes in the subtree rooted at N_{ℓ_j}, which are words of the form $[\star\ell_0 \star \ell_1 \star \cdots \star \ell_j\star]$. Remark that the cofaces of a simplex are represented by a set of subtrees in the simplex tree. The procedure searches only the roots of these subtrees.

The complexity for searching the cofaces of a simplex σ of dimension j depends on the number $\mathcal{T}^{>j}_{last(\sigma)}$ of nodes with label $last(\sigma)$ and depth at least $j+1$. If k is the dimension of the simplicial complex, traversing the tree upwards takes $O(k)$ time. The complexity of this procedure is thus $O(k\mathcal{T}^{>j}_{last(\sigma)})$.

Locate Facets. Locating the facets of a simplex efficiently is the key point of the incremental algorithm we use to construct witness complexes in section 3.2.

Given a simplex σ, we want to access the nodes of the simplex tree representing the facets of σ. If the word representation of σ is $[\ell_0, \cdots, \ell_j]$, the word representations of the facets of σ are the words $[\ell_0, \cdots, \hat{\ell}_i, \cdots, \ell_j]$, $0 \leq i \leq j$, where $\hat{\ell}_i$ indicates that ℓ_i is omitted. If we denote by N_{ℓ_i} the node representing the word $[\ell_0, \cdots, \ell_i]$, a traversal from the node representing σ up to the root will exactly pass through the nodes N_{ℓ_i}, $i = j \cdots 0$. When reaching node $N_{\ell_{i-1}}$, a search from $N_{\ell_{i-1}}$ downwards for the word $[\ell_{i+1}, \cdots, \ell_j]$ will locate (or prove the absence of) the facet $[\ell_0, \cdots, \hat{\ell}_i, \cdots, \ell_j]$. See Figure 2 for a running example.

This procedure locates all the facets of the j-simplex σ in $O(j^2 D_m)$ operations.

Experiments. We present the experimental performance of the facets and cofaces look-up, which are key operations in the construction of witness complexes. Figure 3 represents the average time for these operations on a simplex, as a function of the dimension of the simplex. We use the dataset **Bro**, consisting of points in $\mathbb{R}^{25}$, on top of which we build a relaxed witness complex with 300 landmarks and $15,000$ witnesses, and relaxation parameter $\rho = 0.15$. We obtain a 13-dimensional simplicial complex with $140K$ faces in less than 3 seconds.

The theoretical complexity of computing the facets of a j-simplex σ is $O(j^2 D_m)$. As reported in Figure 3, the average time to search all facets of a j-simplex is well approximated by a quadratic function of the dimension j (the standard error in the approximation is 2.0%).

A bound on the complexity of computing the cofaces of a j-simplex σ is $O(k\mathcal{T}^{>j}_{last(\sigma)})$, where

Dim.Face	0	1	2	3	4	5	6	7	8	9	10	11	12	13
# Faces	300	2700	8057	15906	25271	30180	26568	17618	8900	3445	1015	217	30	2

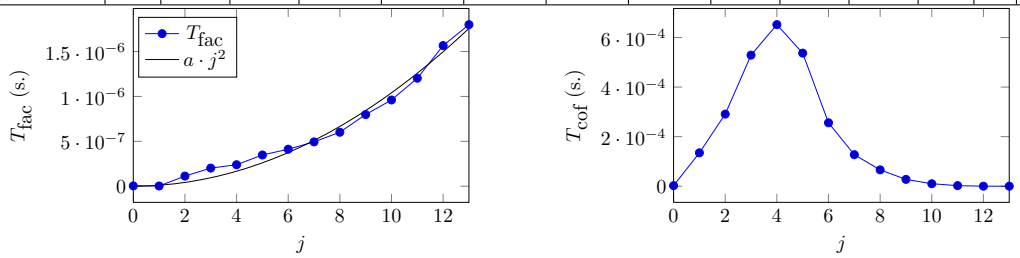


Figure 3: Average time to compute the facets (left) and the cofaces (right) of a simplex of a given dimension.

$\mathcal{T}^{>j}_{last(\sigma)}$ stands for the number of nodes in the simplex tree that store the label $last(\sigma)$ and have depth more than $j+1$. The actual complexity depends on the labelling of the vertices. Figure 3 provides experimental results for a random labelling of the vertices. As can be seen, the time for computing the cofaces of a simplex σ is low, on average, when the dimension of σ is either small (0 to 2) or big (6 to 13), and higher for intermediate dimensions (3 to 5).

2.2.2 Topology preserving operations

We show how to implement two topology preserving operations on a simplicial complex represented as a simplex tree. Such simplifications are, in particular, important in topological data analysis.

Elementary collapse. We say that a simplex σ is collapsible through one of its faces τ if σ is the only coface of τ, which can be checked by computing the cofaces of τ. Such a pair (τ, σ) is called a *free pair*. Removing both faces of a free pair is an elementary collapse.

Since τ has no coface other than σ, the node representing τ in the simplex tree is either a leaf (and so is the node representing σ), or it has the node representing σ as its unique child. An elementary collapse of the free pair (τ, σ) consists either in the removal of the two leaves representing τ and σ, or the removal of the two-nodes subtree containing the nodes representing τ and σ.

Edge contraction. Edge contractions are used in [2] as a tool, under certain conditions, for homology preserving simplification. Let $\mathcal{K}$ be a simplicial complex and let $\{v_{\ell_a}, v_{\ell_b}\}$ be an edge of $\mathcal{K}$ we want to contract. We say that we contract v_{ℓ_b} to v_{ℓ_a} meaning that v_{ℓ_b} is removed from the complex and the link of v_{ℓ_a} is augmented with the link of v_{ℓ_b}. Formally, we define the map f on the set of vertices V which maps v_{ℓ_b} to v_{ℓ_a} and acts as the identity function for all other inputs:

$$f(u) = \begin{cases} v_{\ell_a} & \text{if} \quad u = v_{\ell_b} \\ u & \text{otherwise} \end{cases}$$

We then extend f to all simplices $\sigma = \{v_{\ell_0}, \cdots, v_{\ell_j}\}$ of $\mathcal{K}$ with $f(\sigma) = \{f(v_{\ell_0}), \cdots, f(v_{\ell_j})\}$. The

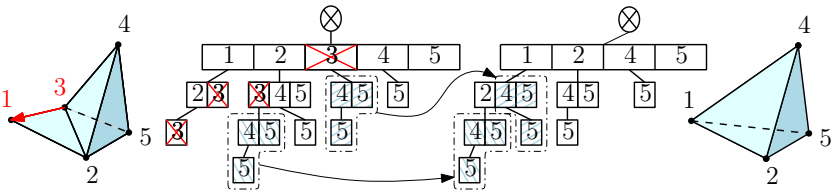


Figure 4: Contraction of vertex 3 to vertex 1 and the associated modifications of the simplicial complex and of the simplex tree. The nodes which are removed are marked with a red cross, the subtrees which are moved are colored in blue.

contraction of v_{ℓ_b} to v_{ℓ_a} is defined as the operation which replaces $\mathcal{K}$ by $\mathcal{K}' = \{f(\sigma)|\sigma \in \mathcal{K}\}$. $\mathcal{K}'$ is a simplicial complex. Let σ be a simplex of $\mathcal{K}$. We distinguish three cases : 1. σ does not contain v_{ℓ_b} and remains unchanged; 2. σ contains both v_{ℓ_a} and v_{ℓ_b}, and $f(\sigma) = \sigma \setminus \{v_{\ell_b}\}$; $|f(\sigma)| = |\sigma| - 1$ and $f(\sigma)$ is a strict subface of σ; 3. σ contains v_{ℓ_b} but not v_{ℓ_a} and $f(\sigma) = (\sigma \setminus \{v_{\ell_b}\}) \cup \{v_{\ell_a}\}$ ($|f(\sigma)| = |\sigma|$).

We describe now how to compute the contraction of v_{ℓ_b} to v_{ℓ_a} when $\mathcal{K}$ is represented as a simplex tree. We suppose that the edge $\{v_{\ell_a}, v_{\ell_b}\}$ is in the complex and, without loss of generality, $\ell_a < \ell_b$. All the simplices which do not contain v_{ℓ_b} remain unchanged and we do not consider them. If a simplex σ contains both v_{ℓ_a} and v_{ℓ_b}, it will become $\sigma \setminus \{v_{\ell_b}\}$ which is a simplex already in $\mathcal{K}$ which does not contain v_{ℓ_b} and will remain in $\mathcal{K}'$. We simply remove σ from the simplex tree. Finally, if σ contains v_{ℓ_b} but not v_{ℓ_a}, we need to remove σ from the simplex tree and add the new simplex $(\sigma \setminus \{v_{\ell_b}\}) \cup \{v_{\ell_a}\}$.

We consider each node N_{ℓ_b} with label ℓ_b in turn. Let σ be the simplex represented by N_{ℓ_b}. The algorithm traverses the tree upwards from N_{ℓ_b} and collects the vertices of σ. Let $T_{N_{\ell_b}}$ be the subtree rooted at N_{ℓ_b}. As $\ell_a < \ell_b$, if σ contains both v_{ℓ_a} and v_{ℓ_b}, this will be true for all the simplices whose representative nodes are in $T_{N_{\ell_b}}$, and, if σ contains only v_{ℓ_b}, the same will be true for all the simplices whose representative nodes are in $T_{N_{\ell_b}}$. Consequently, if σ contains both v_{ℓ_a} and v_{ℓ_b}, we remove the whole subtree $T_{N_{\ell_b}}$ from the simplex tree. Otherwise, σ contains only v_{ℓ_b}, all words represented in $T_{N_{\ell_b}}$ are of the form $[\sigma'] \cdot [\sigma''] \cdot [\ell_b] \cdot [\sigma''']$ and will be turned into words $[\sigma'] \cdot [\ell_a] \cdot [\sigma''] \cdot [\sigma''']$. We then have to move the subtree $T_{N_{\ell_b}}$ (except its root) from position $[\sigma'] \cdot [\sigma'']$ to position $[\sigma'] \cdot [\ell_a] \cdot [\sigma'']$ in the simplex tree. If a subtree is already rooted at this position, we have to merge $T_{N_{\ell_b}}$ with this subtree as illustrated in Figure 4. In order to merge the subtree $T_{N_{\ell_b}}$ with the subtree rooted at the node representing the word $[\sigma'] \cdot [\ell_a] \cdot [\sigma'']$, we can successively insert every node of $T_{N_{\ell_b}}$ is the corresponding set of sibling nodes, stored in a dictionary. This will take in the worst case $O(D_m)$ operations per node in $T_{N_{\ell_b}}$.

We analyse the complexity of contracting an edge $\{v_{\ell_a}, v_{\ell_b}\}$. For each node storing the label ℓ_b, we traverse the tree upwards. This takes $O(k)$ time if the simplicial complex has dimension k. As there are $\mathcal{T}_{\ell_b}^{>0}$ such nodes, the total cost is $O(k\mathcal{T}_{\ell_b}^{>0})$.

We also manipulate the subtrees rooted at the nodes storing label ℓ_b. Specifically, either we remove such a subtree or we move a subtree by changing its parent node. In the latter case, we have to merge two subtrees. This is the more costly operation which takes, in the worst case, $O(D_m)$ operations per node in the subtrees to be merged. As any node in such a subtree represents a coface of vertex v_{ℓ_b}, the total number of nodes in all the subtrees we have to manipulate is at most $C_{\{v_{\ell_b}\}}$, and the manipulation of the subtrees takes $O(C_{\{v_{\ell_b}\}} D_m)$ time. Consequently, the time needed to contract edge $\{v_{\ell_a}, v_{\ell_b}\}$ is $O(k\mathcal{T}_{\ell_b}^{>0} + C_{\{v_{\ell_b}\}} D_m)$.

3 Construction of Simplicial Complexes

In this section, we detail how to implement two important types of simplicial complexes, the flag and the witness complexes, using simplex trees.

3.1 Flag complexes

A flag complex is a simplicial complex whose combinatorial structure is entirely determined by its 1-skeleton. Specifically, a simplex is in the flag complex if and only if its vertices form a clique in the graph of the simplicial complex, or, in other terms, if and only if its vertices are pairwise linked by an edge.

Expansion. Given the 1-skeleton of a flag complex, we call *expansion of order k* the operation which reconstructs the k-skeleton of the flag complex. If the 1-skeleton is stored in a simplex tree, the expansion of order k consists in successively inserting all the simplices of the k-skeleton into the simplex tree.

Let $G = (V, E)$ be the graph of the simplicial complex, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. For a vertex $v_\ell \in V$, we denote by

$$\mathcal{N}^+(v_\ell) = \{\ell' \in \{1, \cdots, |V|\} \mid (v_\ell, v_{\ell'}) \in E \wedge \ell' > \ell\}$$

the set of labels of the neighbors of v_ℓ in G that are bigger than ℓ. Let N_{ℓ_j} be a node in the tree that stores the label ℓ_j and represents the word $[\ell_0, \cdots, \ell_j]$. The children of N_{ℓ_j} store the labels in $\mathcal{N}^+(v_{\ell_0}) \cap \cdots \cap \mathcal{N}^+(v_{\ell_j})$. Indeed, the children of N_{ℓ_j} are neighbors in G of the vertices v_{ℓ_i}, $0 \leq i \leq j$, (by definition of a clique) and must have a bigger label than $\ell_0, \cdots, \ell_j$ (by construction of the simplex tree).

Consequently, the sibling nodes of N_{ℓ_j} are exactly the nodes that store the labels in $A = \mathcal{N}^+(v_{\ell_0}) \cap \cdots \cap \mathcal{N}^+(v_{\ell_{j-1}})$, and the children of N_{ℓ_j} are exactly the nodes that store the labels in $A \cap \mathcal{N}^+(v_{\ell_j})$. See Figure 5.

For every vertex v_ℓ, we have an easy access to $\mathcal{N}^+(v_\ell)$ since $\mathcal{N}^+(v_\ell)$ is exactly the set of labels stored in the children of the top node storing label ℓ. We easily deduce an in-depth expansion algorithm.

The time complexity for the expansion algorithm depends on our ability to compute fast intersections of the type $A \cap \mathcal{N}^+(v_{\ell_j})$. In practice, we have observed that the time taken by the expansion algorithm depends linearly on the size of the output simplicial complex. This means that, on average, the time to compute the intersection of A with $\mathcal{N}^+(v_{\ell_j})$ depends linearly on the size of the output set $A \cap \mathcal{N}^+(v_{\ell_j})$.

Rips Complex. Rips complexes are geometric flag complexes which are popular in computational topology due to their simple construction and their good approximation properties [3]. Given a set of vertices V in a metric space and a parameter $r > 0$, the Rips graph is defined as the graph whose set of vertices is V and two vertices are joined by an edge if their distance is at most r. The Rips complex is the flag complex defined on top of this graph. We will use this complex for our experiments on the construction of flag complexes.

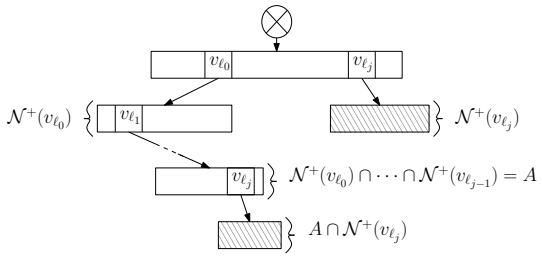


Figure 5: Representation of a set of sibling nodes as intersection of neighborhoods.

3.2 Witness complexes

The Witness Complex has been first introduced in [8]. Its definition involves two given sets of points in a metric space, the set of landmarks L and the set of witnesses W.

Definition 1 *A witness $w \in W$* witnesses *a simplex $\sigma \subseteq L$ iff:*

$$\forall x \in \sigma \text{ and } \forall y \in L \setminus \sigma \text{ we have } d(w,x) \leq d(w,y)$$

or, equivalently, the vertices of σ are the $|\sigma|$ nearest neighbors of w in L.

The *witness complex* $\mathrm{Wit}(W,L)$ is the maximal simplicial complex, with vertex set L, whose faces admit a witness in W. Equivalently, a simplex belongs to the witness complex if and only if it is witnessed and all its facets belong to the witness complex. A simplex satisfying this property will be called *fully witnessed*.

Construction Algorithm. We suppose the sets L and W to be finite and give them labels $\{1,\cdots,|L|\}$ and $\{1,\cdots,|W|\}$ respectively. We describe how to construct the k-skeleton of the witness complex, where k may be any integer in $\{1,\cdots,|L|-1\}$.

Our construction algorithm is incremental, from lower to higher dimensions. If the simplices of the $(j-1)$-skeleton of $\mathrm{Wit}(W,L)$ have been inserted in the simplex tree, we consider all j-dimensional witnessed simplices. For each such simplex σ, we check if its facets are already in the tree and, in the affirmative, we insert σ in the simplex tree.

During the construction of the k-skeleton of the witness complex, we need to access the nearest neighbors of the witnesses, in L. To do so, we compute the $k+1$ nearest neighbors of all the witnesses in a preprocessing phase, and store them in a $|W| \times (k+1)$ matrix. Given an index $j \in \{0,\cdots,k\}$ and a witness $w \in W$, we can then access in constant time the $(j+1)^{\text{th}}$ nearest neighbor of w. We denote this landmark by s_j^w. We maintain a list of *active witnesses*, initialized with W. We insert the vertices of $\mathrm{Wit}(W,L)$ in the simplex tree. For each witness $w \in W$ we insert a top node storing the label of the nearest neighbor of w in L, if no such node already exists. w is initially an active witness and we make it point to the node mentionned above, representing the 0-dimensional simplex w witnesses.

We maintain the following loop invariants: 1. at the beginning of iteration j, the simplex tree contains the $(j-1)$-skeleton of the witness complex $\mathrm{Wit}(W,L)$; 2. the active witnesses are the elements of W that witness a $(j-1)$-simplex of the complex; each active witness w points to the node representing the $(j-1)$-simplex in the tree it witnesses.

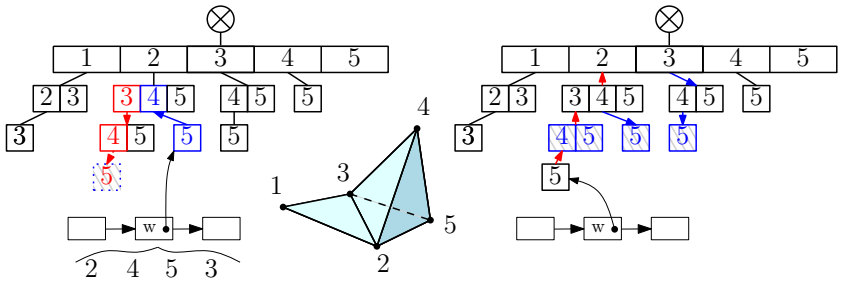


Figure 6: Third iteration of the witness complex construction. The active witness w witnesses the tetrahedron $\{2,3,4,5\}$ and points to the triangle $\{2,4,5\}$. (Left) Search for the potential position of the simplex $\{2,3,4,5\}$ in the simplex tree. (Right) Facets location for simplex $\{2,3,4,5\}$, and update of the pointer of the active witness w.

At iteration $j \geq 1$, we traverse the list of active witnesses. Let w be an active witness. We first compute the $(j+1)^{\text{th}}$ nearest neighbor s_j^w of w using the nearest neighbors matrix (Step 1). Let σ_j be the j-simplex witnessed by w and let us decompose the word representing σ_j into $[\sigma_j] = [\sigma'] \cdot [s_j^w] \cdot [\sigma'']$ ("$\cdot$" denotes the concatenation of words). We then look for the location in the tree where σ_j might be inserted (Step 2). To do so, we start at the node N_w which represents the $(j-1)$-simplex witnessed by w. Observe that the word associated to the path from the root to N_w is exactly $[\sigma'] \cdot [\sigma'']$. We walk $|[\sigma'']|$ steps up from N_w, reach the node representing $[\sigma']$ and then search downwards for the word $[s_w^j] \cdot [\sigma'']$ (see Figure 6, left). The cost of this operation is $O(jD_m)$.

If the node representing σ_j exists, σ_j has already been inserted; we update the pointer of w and return. If the simplex tree contains neither this node nor its father, σ_j is not fully witnessed because the facet represented by its longest prefix is missing. We consequently remove w from the set of active witnesses. Lastly, if the node is not in the tree but its father is, we check whether σ_j is fully witnessed. To do so, we search for the $j+1$ facets of σ_j in the simplex tree (Step 3). The cost of this operation is $O(j^2 D_m)$ using the facets look-up algorithm described in section 2.2. If σ_j is fully witnessed, we insert σ_j in the simplex tree and update the pointer of the active witness w. Else, we remove w from the list of active witnesses (see Figure 6, right).

It is easily seen that the loop invariants are satisfied at the end of iteration j.

Complexity. The cost of accessing a neighbor of a witness using the nearest neighbors matrix is $O(1)$. We access a neighbor (Step 1) and locate a node in the simplex tree (Step 2) at most $k|W|$ times. In total, the cost of Steps 1 and 2 together is $O((kD_m+1)k|W|)$. In Step 3, either we insert a new node in the simplex tree, which happens exactly $|\mathcal{K}|$ times (the number of faces in the complex), or we remove an active witness, which happens at most $|W|$ times. The total cost of Step 3 is thus $O((|\mathcal{K}|+|W|)k^2 D_m)$. In conclusion, constructing the k-skeleton of the witness complex takes time

$$O((|\mathcal{K}|+|W|)k^2 D_m + k|W|) = O((|\mathcal{K}|+|W|)k^2 D_m).$$

Landmark Insertion. We present an algorithm to update the simplex tree under landmark insertions. Adding new vertices is, for example, important for mesh refinement. Given the set

of landmarks L, the set of witnesses W and the k-skeleton of the witness complex $\mathrm{Wit}(W, L)$ represented as a simplex tree, we take a new landmark point x and we update the simplex tree so as to construct the simplex tree associated to $\mathrm{Wit}(W, L \cup \{x\})$. We assign to x the biggest label $|L| + 1$. We suppose to have at our disposal an oracle that can compute the subset $W^x \subseteq W$ of the witnesses that admit x as one of their $k + 1$ nearest neighbors. Computing W^x is known as the *reverse nearest neighbor* search problem, which has been intensively studied in the past few years [1]. Let w be a witness in W^x and suppose x is its $(i + 1)^{\text{th}}$ nearest neighbor in $L \cup \{x\}$, with $0 \leq i \leq k$. Let $\sigma_j \subseteq L$ be the j-dimensional simplex witnessed by w in L and let $\widetilde{\sigma_j} \subseteq L \cup \{x\}$ the j-dimensional simplex witnessed by w in $L \cup \{x\}$. Consequently, $\sigma_j = \widetilde{\sigma_j}$ for $j < i$ and $\sigma_j \neq \widetilde{\sigma_j}$ for $j \geq i$. We equip each node N of the simplex tree with a *counter of witnesses* which maintains the number of witnesses that witness the simplex represented by N. As for the witness complex construction, we consider all nodes representing simplices witnessed by elements of W^x, proceeding by increasing dimensions. For a witness $w \in W^x$ and a dimension $j > i$, we decrement the witness counter of σ_j and insert $\widetilde{\sigma_j}$ if and only if its facets are in the simplex tree. We remark that $[\widetilde{\sigma_j}] = [\sigma_{j-1}] \cdot [x]$ because x has the biggest label of all landmarks. We can thus access in time $O(D_m)$ the position of the word $[\widetilde{\sigma_j}]$ since we have accessed the node representing $[\sigma_{j-1}]$ in the previous iteration of the algorithm.

Complexity. The update procedure is a "local" variant of the witness complex construction, where, by "local", we mean that we reconstruct only the star of vertex x. Let C_x denote the number of cofaces of x in $\mathrm{Wit}(W, L \cup \{x\})$ (or equivalently the size of its star). The same analysis as above shows that updating the simplicial complex takes time $O((|W^x| + C_x)k^2 D_m)$, plus one call to the oracle to compute W^x.

Relaxed Witness Complex. Given a relaxation parameter $\rho \geq 0$ we define the *relaxed witness complex* [8]:

Definition 2 *A witness $w \in W$* ρ-witnesses *a simplex $\sigma \subseteq L$ iff:*

$$\forall x \in \sigma \text{ and } \forall y \in L \setminus \sigma \text{ we have } d(w, x) \leq d(w, y) + \rho$$

The *relaxed witness complex* $\mathrm{Wit}^\rho(W, L)$ with parameter ρ is the maximal simplicial complex, with vertex set L, whose faces admit a ρ-witness in W. For $\rho = 0$, the relaxed witness complex is the standard witness complex. The parameter ρ defines a filtration on the witness complex, which has been used in topological data analysis.

We resort to the same incremental algorithm as above. At each step j, we insert, for each witness w, the j-dimensional simplices which are ρ-witnessed by w. Differently from the standard witness complex, there may be more than one j-simplices that are witnessed by a given witness $w \in W$. Consequently, we do not maintain a pointer from each active witness to the last inserted simplex it witnesses. We use simple top-down insertions from the root of the simplex tree.

Given a witness w and a dimension j, we generate all the j-dimensional simplices which are ρ-witnessed by w. For the ease of exposition, we suppose we are given the sorted list of nearest neighbors of w in L, noted $\{z_0 \cdots z_{|L|-1}\}$, and their distance to w, noted $m_i = \mathrm{d}(w, z_i)$, with $m_0 \leq \cdots \leq m_{|L|-1}$, breaking ties arbitrarily. Note that if one wants to construct only the k-skeleton of the complex, it is sufficient to know the list of neighbors of w that are at distance at most $m_k + \rho$ from w. We preprocess this list of neighbors for all witnesses. For $i \in \{0, \cdots, |L|-1\}$,

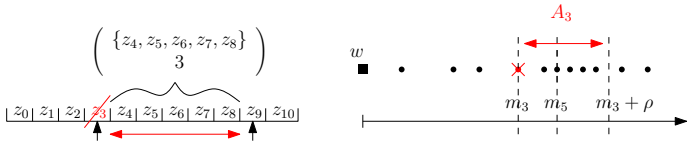


Figure 7: Computation of the ρ-witnessed simplices σ of dimension 5. If z_3 is the first neighbor of w not in σ, then σ contains $\{z_0, z_1, z_2\}$ and any 3-uplet of $A_3 = \{z_4, \cdots, z_8\}$.

we define the set A_i of landmarks z such that $m_i \leq d(w,z) \leq m_i + \rho$. For $i \leq j+1$, w ρ-witnesses all the j-simplices that contain $\{z_0, \cdots, z_{i-1}\}$ and a $(j+1-i)$-subset of A_i, provided $|A_i| \geq j+1-i$. It is easy to see that all j-simplices that are ρ-witnessed by w are obtained this way, and exactly once, when i ranges from 0 to $j+1$.

For all $i \in \{0, \cdots, j+1\}$, we compute A_i and generate all the simplices which contain $\{z_0, \cdots, z_{i-1}\}$ and a subset of A_i of size $(j+1-i)$. In order to easily update a_i when i is incremented, we maintain two pointers to the list of neighbors, one to z_i and the other to the end of A_i. We check in constant time if A_i contains more than $j+1-i$ vertices, and compute all the subsets of A_i of cardinality $j+1-i$ accordingly. See Figure 7.

Complexity. Let R_j be the number of j-simplices ρ-witnessed by w. Generating all those simplices takes $O(j+R_j)$ time. Indeed, for all i from 0 to $j+1$, we construct A_i and check whether A_i contains more than $j+1-i$ elements. This is done by a simple traversal of the list of neighbors of w, which takes $O(j)$ time. Then, when A_i contains more than $j+1-i$ elements, we generate all subsets of A_i of size $j+1-i$ in time $O(\binom{|A_i|}{j+1-i})$. As each such subset will lead to a ρ-witnessed simplex, the total cost for generating all those simplices is $O(R_j)$.

We can deduce the complexity of the construction of the relaxed witness complex. Let $\mathcal{R} = \sum_{w \in W} \sum_{j=0 \cdots k} R_j$ be the number of simplices witnessed by elements of W. The construction of the relaxed witness complex takes $O(\mathcal{R} k^2 D_m)$ operations. This bound is quite pessimistic and, in practice, we observed that the construction time is sensitive to the size of the output complex. Observe that the quantity analogous to $\mathcal{R}$ in the case of the standard witness complex was $k|W|$ and that the complexity was better due to our use of the notion of active witnesses.

4 Experiments

In this section, we report on the performance of our algorithms on both real and synthetic data, and compare them to existing software. More specifically, we benchmark the construction of Rips complexes, witness complexes and relaxed witness complexes. Our implementations are in `C++`. We use the `ANN` library [15] to compute the 1-skeleton graph of the Rips complex, and to compute the lists of nearest neighbors of the witnesses for the witness complexes. All timings are measured on a Linux machine with 3.00 GHz processor and 32 GB RAM. Timings are provided by the `clock` function from the `Standard C Library`, and zero means that the measured time is below the resolution of the `clock` function. For its efficiency and flexibility, we use the `map` structure from the `Standard Template Library` [17] for storing sets of sibling nodes, except for the top nodes which are stored in an array.

Data	$\lvert\mathcal{P}\rvert$	D	d	r	k	T_{g}	$\lvert E\rvert$	T_{Rips}	$\lvert\mathcal{K}\rvert$	T_{tot}	$T_{\text{tot}}/\lvert\mathcal{K}\rvert$
Bud	49,990	3	2	0.11	3	1.5	1,275,930	104.5	354,695,000	104.6	$3.0 \cdot 10^{-7}$
Bro	15,000	25	?	0.019	25	0.6	3083	36.5	116,743,000	37.1	$3.2 \cdot 10^{-7}$
Cy8	6,040	24	2	0.4	24	0.11	76,657	4.5	13,379,500	4.61	$3.4 \cdot 10^{-7}$
Kl	90,000	5	2	0.075	5	0.46	1,120,000	68.1	233,557,000	68.5	$2.9 \cdot 10^{-7}$
S4	50,000	5	4	0.28	5	2.2	1,422,490	95.1	275,126,000	97.3	$3.6 \cdot 10^{-7}$

Data	$\lvert L\rvert$	$\lvert W\rvert$	D	d	ρ	k	T_{nn}	T_{Wit^ρ}	$\lvert\mathcal{K}\rvert$	T_{tot}	$T_{\text{tot}}/\lvert\mathcal{K}\rvert$
Bud	10,000	49,990	3	2	0.12	3	1.	729.6	125,669,000	730.6	$12 \cdot 10^{-3}$
Bro	3,000	15,000	25	?	0.01	25	9.9	107.6	2,589,860	117.5	$6.5 \cdot 10^{-3}$
Cy8	800	6,040	24	2	0.23	24	0.38	161	997,344	161.2	$23 \cdot 10^{-3}$
Kl	10,000	90,000	5	2	0.11	5	2.2	572	109,094,000	574.2	$5.7 \cdot 10^{-3}$
S4	50,000	200,000	5	4	0.06	5	25.1	296.7	163,455,000	321.8	$1.2 \cdot 10^{-3}$

Figure 8: Data, timings (in s.) and statistics for the construction of Rips complexes (TOP) and relaxed witness complexes (BOTTOM).

We use a variety of both real and synthetic datasets. **Bud** is a set of points sampled from the surface of the *Stanford Buddha* in $\mathbb{R}^3$. **Bro** is a set of 5×5 *high-contrast patches* derived from natural images, interpreted as vectors in $\mathbb{R}^{25}$, from the Brown database (with parameter $k = 300$ and cut 30%) [12, 6]. **Cy8** is a set of points in $\mathbb{R}^{24}$, sampled from the space of conformations of the cyclo-octane molecule [14], which is the union of two intersecting surfaces. **Kl** is a set of points sampled from the surface of the figure eight Klein Bottle embedded in $\mathbb{R}^5$. Finally **S4** is a set of points uniformly distributed on the unit 4-sphere in $\mathbb{R}^5$. Datasets are listed in Figure 8 with details on the sets of points $\mathcal{P}$ or landmarks L and witnesses W, their size $|\mathcal{P}|$ or $|L|$ and $|W|$, the ambient dimension D, the intrinsic dimension d of the object the sample points belong to (if known), the parameter r or ρ, the dimension k up to which we construct the complexes, the time T_{g} to construct the Rips graph or the time T_{nn} to compute the lists of nearest neighbors of the witnesses, the number of edges $|E|$, the time for the construction of the Rips complex T_{Rips} or for the construction of the witness complex T_{Wit^ρ}, the size of the complex $|\mathcal{K}|$, and the total construction time T_{tot} and average construction time per face $T_{\text{tot}}/|\mathcal{K}|$.

We test the performance of our algorithms on these datasets, and compare them to the `JPlex library` [16] which is a Java software package which can be used with `Matlab`. `JPlex` is widely used to construct simplicial complexes and to compute their homology. We also provide an experimental analysis of the memory performance of our data structure compared to other representations. Unless mentioned otherwise, all simplicial complexes are computed up to the embedding dimension, because the homology is trivial in dimenson higher than the ambient dimension. All timings are averaged over 10 independent runs. Due to the lack of space, we cannot report on the performance of each algorithm on each dataset but the results presented are a faithful sample of what we have observed on other datasets.

As illustrated in Figure 8, we are able to construct and represent both Rips and relaxed witness complexes of up to several hundred million faces in high dimensions, on all datasets.

4.1 Memory Performance of the Simplex Tree

In order to represent the combinatorial structure of an arbitrary simplicial complex, one needs to mark all maximal faces. Indeed, from the definition of a simplicial complex, we cannot infer the higher dimensional faces from the lower dimensional ones. Moreover, the number of maximal

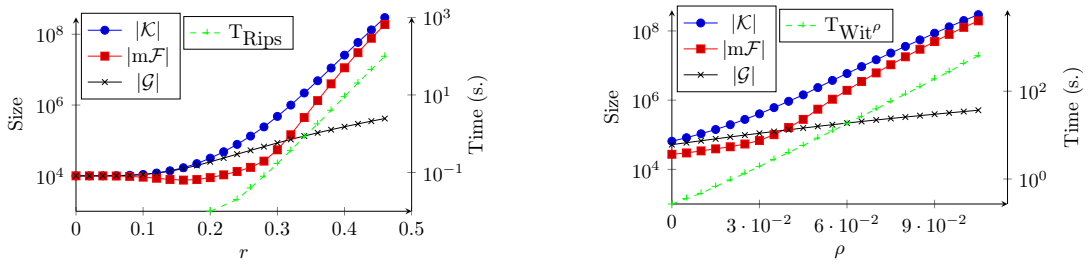


Figure 9: Statistics and timings for the Rips complex (Left) and the relaxed witness complex (Right) on **S4**.

simplices of a k-dimensional simplicial complex is at least $|V|/(k+1)$. In the case, considered in this paper, where the vertices are identified by their labels, a minimal representation of the maximal simplices would then require at least $\Omega(\log|V|)$ bits per maximal face. The simplex tree uses $O(\log|V|)$ memory bits per face *of any dimension*. The following experiment compares the memory performance of the simplex tree with the minimal representation described above, and with the representation of the 1-skeleton only.

Figure 9 shows results for both Rips and relaxed witness complexes associated to $10,000$ points from **S4** and various values of, respectively, the distance threshold r and the relaxation parameter ρ. The figure plots the total number of faces $|\mathcal{K}|$, the number of maximal faces $|\mathrm{m}\mathcal{F}|$, the size of the 1-skeleton $|\mathcal{G}|$ and the construction times T_{Rips} and T_{Wit^ρ}. The quantities $|\mathcal{K}|$, $|\mathrm{m}\mathcal{F}|$ and $|\mathcal{G}|$ stand, respectively, for the asymptotic size of the simplex tree, the asymptotic size of the optimal representation and of the size of the representation of the 1-skeleton.

As expected, the 1-skeleton is significantly smaller than the two other representations. However, as explained earlier, a representation of the graph of the simplicial complex is only well suited for flag complexes.

As shown on the figure, the total number of faces and the number of maximal faces remain close along the experiment. Interestingly, we catch the topology of **S4** when $r \approx 0.4$ for the Rips complex and $\rho \approx 0.08$ for the relaxed witness complex. For these "good" values of the parameters, the total number of faces is not much bigger than the number of maximal faces. Specifically, the total number of faces of the Rips complex is less than 2.3 times bigger than the number of maximal faces, and the ratio is less than 2 for the relaxed witness complex.

4.2 Construction of Rips Complexes

We test our algorithm for the construction of Rips complexes. In Figure 10 we compare the performance of our algorithm with `JPlex` along two directions. In the first experiment, we build the Rips complex on the dataset **Bud**. Our construction is between 200 and 20 times faster (from small to big r) than `JPlex`. Moreover, `JPlex` is not able to handle big simplicial complexes due to memory allocation issues whereas our method has no such problem. In our experiments, `JPlex` is not able to compute complexes of more than $306,000$ faces ($r = 0.12$) while the simplex tree construction runs successfully until $r = 0.5$, resulting in a complex of 415 million faces.

In the second experiment, we construct the Rips complex of **Cy8** for different dimensions k. Again, our method outperforms `JPlex`, by a factor 14 to 20 (from small to big k). Again, `JPlex` cannot compute complexes of dimension higher than 7.

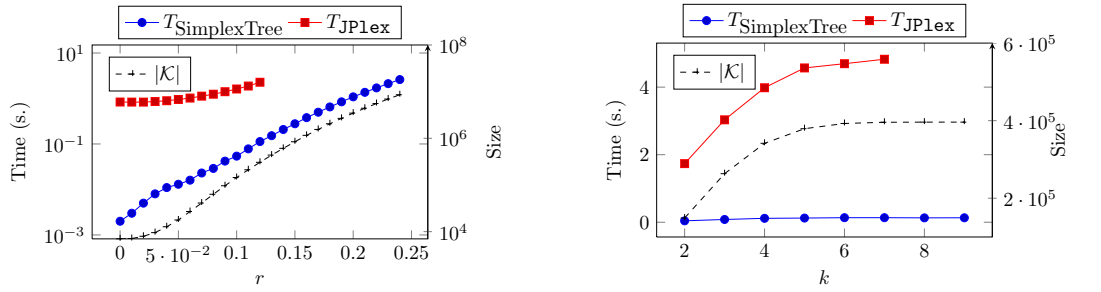


Figure 10: Comparison with `JPlex` on the full construction of the Rips complex. (Left) On a subsample of 7023 points from **Bud** with increasing threshold r. (Right) On **Cy8** dataset for $r = 0.31$ and up to different dimensions k. `JPlex` was not able to process the full dataset **Bud** due to problems in memory allocation, and cannot process in dimensions higher than 7.

The simplex tree and the expansion algorithm we have described are, by design, output sensitive. For example, when we construct the Rips complex on **Cy8** for dimensions k higher than 5, the size of the output complex is constant, and so is the time for the Rips complex construction using a simplex tree. This is not the case for `JPlex`. Even further, the construction time using a simplex tree depends, experimentally, linearly on the size of the output complex. Indeed, when the Rips graphs are dense enough so that the time for the expansion dominates the full construction, we observe that the average construction time per face is constant and equal to 2.9×10^{-7} seconds for the first experiment, and 5.4×10^{-7} seconds for the second experiment (the standard errors around these average times are respectively 0.1% and 1.3%).

4.3 Construction of Witness Complexes

`JPlex` does not provide an implementation of the relaxed witness complex so we were only able to compare the algorithms on the construction of the witness complex. Figure 11 (top) shows the results of two experiments on the full construction of the witness complex. The first one compares the performance of the simplex tree algorithm and of `JPlex` on the dataset **Bro** consisting of $15,000$ points in dimension $\mathbb{R}^{25}$. The landmarks are selected at random among the sample points. Our algorithm is between 14 to 315 times faster (from small to big sets of landmarks) than `JPlex`. Moreover, due to memory allocation issues, `JPlex` is not able to compute the witness complex when selecting more than 6% sample points as landmarks. Differently, our method can compute the witness complex for any number of landmarks. In this experiment, the construction time per face is constant and approximatively equal to 8.5×10^{-5} seconds per face (with standard error 0.8%). This corresponds to the theoretical complexity analysis giving a construction time in $O((|\mathcal{K}| + |W|)k^2 D_m)$ operations, which is linear in $|\mathcal{K}|$ for fixed $|W|$ and fixed k.

In the second experiment, we vary the size of the witness sets. The size of the complex grows very slowly as a function of $|W|$. The construction time of both methods, using `JPlex` and the simplex tree, depends linearly on the number of witnesses. However, the simplex tree algorithm is way more efficient and is about 500 times faster on this experiment. This linear dependency corresponds to the complexity bound $O((|\mathcal{K}| + |W|)k^2 D_m)$ because k is fixed and the size of the complex is small compared to the number of witnesses.

Finally we test the full construction of the relaxed witness complex along two directions, as illustrated in Figure 11 (bottom). In the first experiment, we compute the 5-skeleton of the relaxed witness complex on **Bro** for different values of the parameter ρ. In the second experiment,

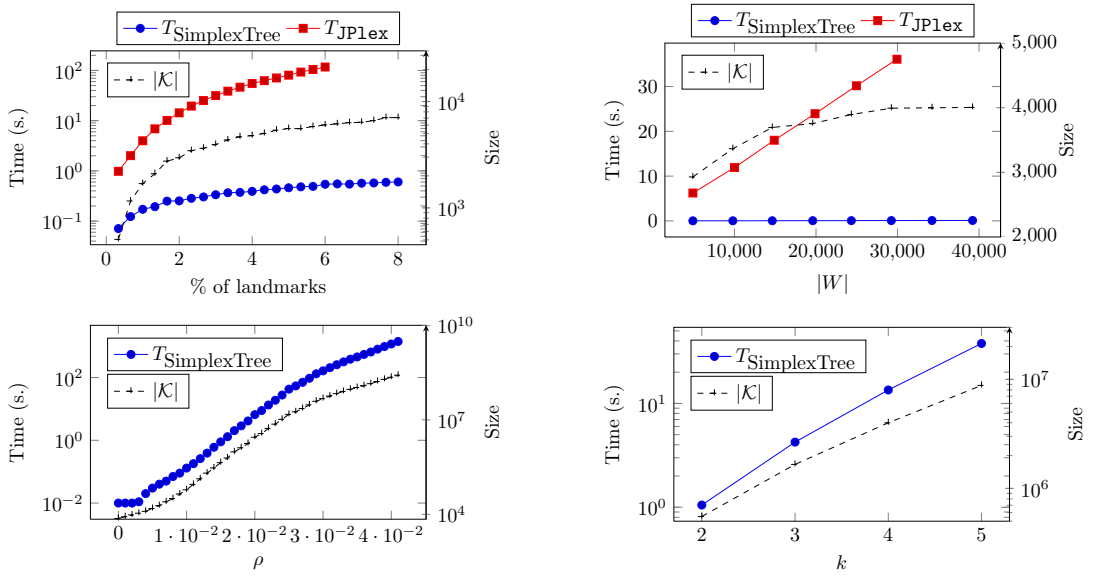


Figure 11: TOP: Comparison with `JPlex` for the full construction of the witness complex. (Left) On **Bro** with different sizes of landmark sets. (Right) On **Kl** with 900 landmarks and different densities of witnesses. BOTTOM: Performance of the Simplex Tree algorithm for the full construction of the relaxed witness complex. (Left) On **Bro** with $1,000$ landmarks and $15,000$ witnesses with different ρ. (Right) On **Kl** with $10,000$ landmarks and $100,000$ witnesses, fixed $\rho = 0.07$ and up to different dimensions k.

we construct the k-skeleton of the relaxed witness complex on **Kl** with fixed parameter ρ and various k. We are able to construct and store complexes of up to 260 million faces. In both cases the construction time is linear in the size of the output complex, with a contruction time per face equal to 4.9×10^{-6} seconds in the first experiment, and 4.0×10^{-6} seconds in the second experiment (the standard errors around these average times are respectively 1.6% and 6.3%).

Conclusion

We believe that the simplex tree is the first scalable and truly practical data structure to represent general simplicial complexes. We plan to integrate our code in the CGAL library and to use it for practical applications in data analysis and manifold learning. Further developments also include more compact storage using succinct representations of trees [11].

Acknowledgments

The authors thanks A.Ghosh, S. Hornus, D. Morozov and P. Skraba for discussions that led to the idea of representing simplicial complexes by tries [10]. They also thank S. Martin and V. Coutsias for providing the cyclo-octane data set.

References

[1] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *IN SIGMOD*, pages 515–526, 2006.

[2] D. Attali, A. Lieutier, and D. Salinas. Efficient data structure for representing and simplifying simplicial complexes in high dimensions. In *Proceedings of the 27th annual ACM symposium on Computational geometry*, SoCG '11, pages 501–509, New York, NY, USA, 2011. ACM.

[3] D. Attali, A. Lieutier, and D. Salinas. Vietoris-rips complexes also provide topologically correct reconstructions of sampled shapes. In *Symposium on Computational Geometry*, pages 491–500, 2011.

[4] J. L. Bentley and R. Sedgewick. Fast algorithms for sorting and searching strings. In *SODA*, pages 360–369, 1997.

[5] E. Brisson. Representing geometric structures in d dimensions: topology and order. In *Proceedings of the fifth annual symposium on Computational geometry*, SCG '89, pages 218–227, New York, NY, USA, 1989. ACM.

[6] G. Carlsson, T. Ishkhanov, V. Silva, and A. Zomorodian. On the local behavior of spaces of natural images. *Int. J. Comput. Vision*, 76:1–12, January 2008.

[7] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 537–546, New York, NY, USA, 2008. ACM.

[8] V. de Silva and G. Carlsson. Topological estimation using witness complexes. In M. Alexa and S. Rusinkiewicz, editors, *Eurographics Symposium on Point-Based Graphics*. The Eurographics Association, 2004.

[9] H. Edelsbrunner and J. Harer. *Computational Topology - an Introduction*. American Mathematical Society, 2010.

[10] S. Hornus. Private communication.

[11] G. Jacobson. Space-efficient static trees and graphs. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:549–554, 1989.

[12] A. B. Lee, K. S. Pedersen, and D. Mumford. The nonlinear statistics of high-contrast patches in natural images. *International Journal of Computer Vision*, 54(1-3):83–103, 2003.

[13] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *Int. J. Comput. Geometry Appl.*, 4(3):275–324, 1994.

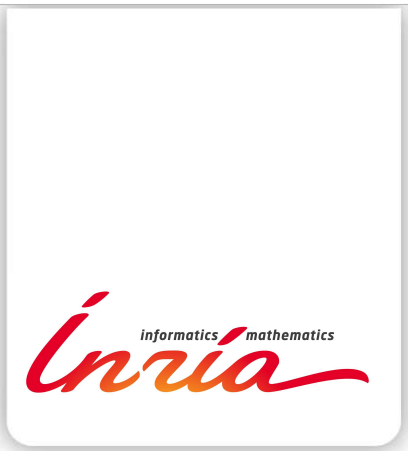
[14] S. Martin, A. Thompson, E. Coutsias, and J. Watson. Topology of cyclo-octane energy landscape. *J Chem Phys*, 132(23):234115, 2010.

[15] D. M. Mount and S. Arya. Ann: A library for approximate nearest neighbor searching version 1.1.2. 2010.

[16] H. Sexton and M. V. Johansson. Jplex. In *http://comptop.stanford.edu/programs/jplex/*, 2009.

[17] SGI. Standard template library programmer's guide. In *http://www.sgi.com/tech/stl/*.

[18] A. Zomorodian. The tidy set: a minimal simplicial set for computing homology of clique complexes. In *Symposium on Computational Geometry*, pages 257–266, 2010.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399