Building Efficient and Compact Data Structures for Simplicial Complexes

Jean-Daniel Boissonnat[*1], **Karthik C. S.**[†2], **and Sébastien Tavenas**[‡3]

1 **Geometrica, INRIA Sophia Antipolis – Méditerranée, France**
Jean-Daniel.Boissonnat@inria.fr
2 **Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Israel**
karthik.srikanta@weizmann.ac.il
3 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
stavenas@mpi-inf.mpg.de

Abstract

The Simplex Tree (ST) is a recently introduced data structure that can represent abstract simplicial complexes of any dimension and allows efficient implementation of a large range of basic operations on simplicial complexes. In this paper, we show how to optimally compress the Simplex Tree while retaining its functionalities. In addition, we propose two new data structures called Maximal Simplex Tree (MxST) and Simplex Array List (SAL). We analyze the compressed Simplex Tree, the Maximal Simplex Tree, and the Simplex Array List under various settings.

1998 ACM Subject Classification E.1 Data structures, F.2.2 Nonnumerical Algorithms and Problems – Computations on discrete structures, Geometrical problems and computations

Keywords and phrases Simplicial complex, Compact data structures, Automaton, NP-hard

Digital Object Identifier 10.4230/LIPIcs.SOCG.2015.642

1 Introduction

Simplicial complexes are widely used in combinatorial and computational topology, and have found many applications in topological data analysis and geometric inference. The most common representation uses the Hasse diagram of the complex that has one node per simplex and an edge between any pair of incident simplices whose dimensions differ by one. A few attempts to obtain more compact representations have been reported recently.

Attali et al. [4] proposed the skeleton-blockers data structure which represents a simplicial complex by its 1-skeleton together with its set of blockers. Blockers are the simplices which are not contained in the complex but whose proper subfaces are. Flag complexes have no blockers and the skeleton-blocker representation is especially efficient for complexes that are "close" to flag complexes. An interesting property of the skeleton-blocker representation is that it enables efficient edge contraction.

Boissonnat and Maria [8] have proposed a tree representation called the Simplex Tree that can represent general simplicial complexes and scales well with dimension. The nodes

* This work was partially supported by the Advanced Grant of the European Research Council GUDHI.
† This work was partially supported by Labex UCN@Sophia scholarship, LIP fellowship and Irit Dinur's ERC-StG grant number 239985.
‡ A part of this work was done at LIP, ENS Lyon (UMR 5668 ENS Lyon – CNRS – UCBL – INRIA, Université de Lyon).

31st International Symposium on Computational Geometry (SoCG'15).
Editors: Lars Arge and János Pach; pp. 642–657

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of the tree are in bijection with the simplices (of all dimensions) of the simplicial complex. In this way, the Simplex Tree explicitly stores all the simplices of the complex but it doesn't represent explicitly all the incidences between simplices that are stored in the Hasse diagram. Storing all the simplices is useful (for example, one can then attach information to each simplex or store a filtration succinctly). Moreover, the tree structure of the Simplex Tree leads to efficient implementation of basic operations on simplicial complexes (such as retrieving incidence relations, and in particular retrieving the faces or the cofaces of a simplex).

In this paper, we propose a way to compress the Simplex Tree so as to store as few nodes and edges as possible without compromising the functionality of the data structure. The new compressed data structure is in fact a finite automaton (referred to in this paper as the Minimal Simplex Automaton) and we describe an optimal algorithm for its construction.

Previous works have looked at trie compression and have tried to establish a good trade-off between speed and size, but in most of the works, the emphasis is on one of the two. Two examples of work where the speed is of main concern are [2] where the query time is improved by reducing the number of levels in a binary trie (which corresponds to truncating the Simplex Tree at a certain height) and [1] where trie data structures are optimized for computer memory architectures. When the size of the structure is of primary concern, the work is usually focused on automata compression. For instance, in the context of natural language data processing, significant savings in memory space can be obtained if the dictionary is stored in a directed acyclic word graph (DAWG), a form of a minimal deterministic automaton, where common suffixes are shared [3]. However, theoretical analysis of compression is seldom done (if at all), in any of these works.

In this paper, we analyze the size of the Minimal Simplex Automaton and also demonstrate (through experiments) that compression works especially well for Simplex Tree due to the structure of simplicial complexes: namely, that all subfaces of a simplex in the complex also belong to the complex. Additionally, we consider the influence of the labeling of the vertices on compression, which can be significant. Further, we show that it is hard to find an optimal labeling for the compressed Simplex Tree and for the Minimal Simplex Automaton.

Further, we introduce two new data structures for simplicial complexes called the Maximal Simplex Tree (MxST) and the Simplex Array List (SAL). MxST is a subtree of the Simplex Tree whose leaves are in bijection with the maximal simplices (i.e. simplices with no cofaces) of the complex. We show that MxST is compact and allows efficient operations. MxST is then augmented to obtain SAL where every node uniquely represents an edge. A nice feature of SAL is its invariance over labeling of vertices. We show that SAL supports efficient basic operations and that it is compact when the dimension of the complex is fixed, a case of great interest in Manifold Learning and Topological Data Analysis. Complete proofs and more detailed discussions are presented in the full version of the paper [7].

2 Simplicial Complex: Definitions and a Lower Bound

In this paper, the class of d dimensional simplicial complexes on n vertices with m simplices, of which k are maximal, is denoted by $\mathcal{K}(n,k,d,m)$, and K denotes a simplicial complex in $\mathcal{K}(n,k,d,m)$. At times, we say $K_\theta \in \mathcal{K}_\theta(n,k,d,m)$ (where $\theta : V \to \{1,2,...,|V|\}$ is a labeling of the vertex set V of K) when we want to emphasize that some of the data structures seen in this paper are influenced by the labeling of the vertices.

A maximal simplex of a simplicial complex is a simplex which is not contained in a larger simplex of the complex. A simplicial complex is pure, if all its maximal simplices are of the same dimension. Also, a free pair is defined as a pair of simplices (τ,σ) in K where τ is the

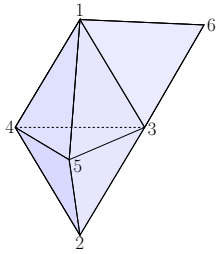

Figure 1 Simplicial complex with the tetrahedra 1–3–4–5 and 2–3–4–5, and the triangle 1–3–6.

only coface of σ. In Figure 1, we have a simplicial complex on vertex set $\{1, 2, 3, 4, 5, 6\}$ which has three maximal simplices: the two tetrahedra 1–3–4–5 and 2–3–4–5, and the triangle 1–3–6. We use this complex as an example through out the paper.

We would like to note here that the case when $k = \mathcal{O}(n)$, is of particular interest. It can be observed in flag complexes, constructed from planar graphs and expanders [13], and in general, from nowhere dense graphs [15], and also from chordal graphs[14]. Generalizing, for all flag complexes constructed from graphs with degeneracy $\mathcal{O}(\log n)$ (degeneracy is the smallest integer r such that every subgraph has a vertex of degree at most r), we have that $k = n^{\mathcal{O}(1)}$ [13]. This encompasses a large class of complexes encountered in practice.

Now, we obtain a lower bound on the space needed to represent simplicial complexes by presenting a counting argument on the number of distinct simplicial complexes.

▶ **Theorem 1.** *Consider the set of all simplicial complexes $\mathcal{K}(n, k, d, m)$ where $d \geq 2$ and $k \geq n + 1$, and consider any data structure that can represent the simplicial complexes of this class. Such a data structure requires $\log \binom{\binom{n/2}{d+1}}{k-n}$ bits to be stored. For any constant $\varepsilon \in (0, 1)$ and for $\frac{2}{\varepsilon} n \leq k \leq n^{(1-\varepsilon)d}$ and $d \leq n^{\varepsilon/3}$, the bound becomes $\Omega(kd \log n)$.*

Proof Sketch. Define $h = k - n \geq 1$ and suppose there exists a data structure that is stored only using $s < \log \alpha \stackrel{\text{def}}{=} \log \binom{\binom{n/2}{d+1}}{h}$ bits. We will construct α simplicial complexes, all with the same set P of n vertices, the same dimension d, and with exactly k maximal simplices. Then, we will have two different complexes, say K and K', encoded by the same word. But, by the construction of these complexes, there is a simplex which is in K and not in K'. ◀

Theorem 1 applies particularly to the case of pseudomanifolds of fixed dimension where we have $k \leq n^{\frac{d}{2}}$ (i.e. $\varepsilon = \frac{1}{2}$ suffices) [6]. The case where d is small is important in Manifold Learning where it is usually assumed that the data live close to a manifold of small intrinsic dimension. The dimension of the simplicial complex should reflect this fact and ideally be equal to the dimension of the manifold.

3 Compression of the Simplex Tree

Let $K \in \mathcal{K}(n, k, d, m)$ be a simplicial complex whose vertices are labeled from 1 to n and ordered accordingly. We can thus associate to each simplex of K a word on the alphabet set $\{1, \ldots, n\}$. Specifically, a j-simplex of 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 $\sigma = \{v_{\ell_0}, \ldots, v_{\ell_j}\}$ be a simplex, where v_{ℓ_i} are vertices of K and $\ell_i \in \{1, \ldots, n\}$ and $\ell_0 < \cdots < \ell_j$. σ is represented by the word $[\sigma] = [\ell_0, \cdots, \ell_j]$. The simplicial complex K can be defined as a collection of words on an alphabet of size n. To compactly represent the set of simplices of K, the corresponding words are stored in a tree and this data structure is called the Simplex

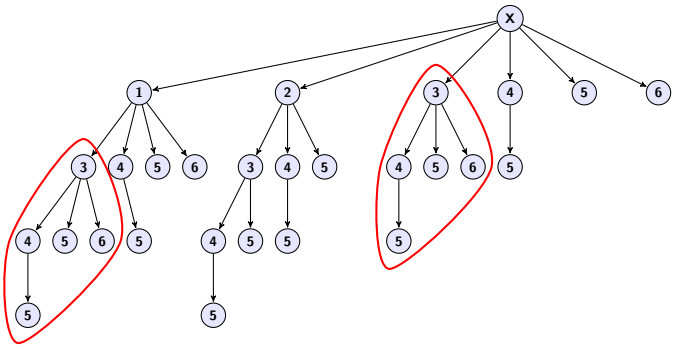

Figure 2 Simplex Tree of the simplicial complex in Figure 1.

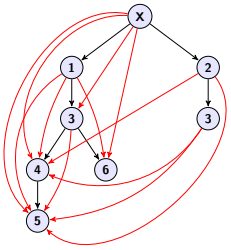

Figure 3 Compressed Simplex Tree of the Simplex Tree given in Figure 2.

Tree of K and denoted by $\mathrm{ST}(K)$ or simply ST when there is no ambiguity. It may be seen as a trie on the words representing the simplices of the complex. 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.

We give a constructive definition of ST. Starting from an empty tree, insert the words representing the simplices of the complex in the following manner. When inserting the word $[\sigma] = [\ell_0, \cdots, \ell_j]$ 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 ST. Next, append to the node representing $[\ell_0, \cdots, \ell_i]$ a path consisting of the nodes storing labels $\ell_{i+1}, \cdots, \ell_j$. In Figure 2, we give ST for the simplicial complex shown in Figure 1.

If K consists of m simplices (including the empty face) then, the associated ST contains exactly m nodes. Thus, we need $\Theta(m \log n)$ space/bits to represent ST (since each node stores a vertex which needs $\Theta(\log n)$ bits to be represented). We can compare this to the lower bound of Theorem 1. In particular, if $k = \mathcal{O}(1)$ then, ST requires at least $\Omega(2^d \log n)$ bits where as Theorem 1 proves the necessity of only $\Omega(d \log n)$ bits. Therefore, while ST is an efficient data structure for some basic operations such as determining membership of a simplex and computing the r-skeleton of the complex, it requires storing every simplex explicitly through a node, leading to combinatorial redundancy. To overcome this, we compress ST as described below.

3.1 Compressed Simplex Tree

Consider the ST in Figure 2 and note that the parts marked in red appear twice. The goal of the compression is to identify these common parts and store them only once. More concretely, if the same subtree is rooted at two different nodes in ST, then the subtree is stored only once and the two root nodes now point to the unique copy of the subtree. As a consequence, the nodes are no longer in bijection with the nodes of the complex (as it was in the case of ST), but we still have the property that the paths from the root are in bijection with the simplices. We see in Figure 3, the compressed ST of the simplicial complex described in Figure 1. In the rest of the paper, we denote by $\mathcal{C}$, this action of compression. Also, unless otherwise stated, $|\mathrm{ST}|$ and $|\mathcal{C}(\mathrm{ST})|$ refer to the number of edges in ST and $\mathcal{C}(\mathrm{ST})$ respectively.

Answering simplex membership queries and other queries that only require traversing the ST from root to leaves can be implemented in $\mathcal{C}(\mathrm{ST})$ exactly as in ST [8]. Allowing upward traversal in ST is also possible (with additional pointers from children to parents) and this has been shown to improve the efficiency of some operations, such as face or coface retrieval. In $\mathcal{C}(\mathrm{ST})$, parents are not unique. To account for this, we mark the parents that

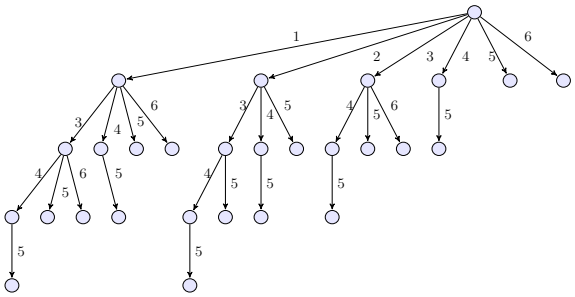

Figure 4 Simplex Automaton of the complex in Figure 1.

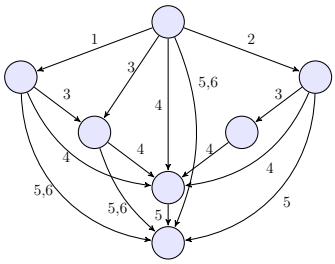

Figure 5 Minimal Simplex Automaton of the complex in Figure 1.

were accessed, and use this to go back in the upward direction. This implies an additional storage of $\mathcal{O}(d \log n)$ while traversing, but a node (simplex) having many parents can assist to locate cofaces much faster.

Next, we will introduce an automaton perspective of the above compression and show how to deduce the optimal compression algorithm for the ST. We will also describe insertion and removal operations on $\mathcal{C}(\text{ST})$ through the automaton perspective.

3.2 Minimal Simplex Automaton

A Deterministic Finite state Automaton (DFA) recognizing a language is defined by a set of states and labeled transitions between these states to detect if a given word is in a predefined language or not. ST can be seen as a DFA: let us define the set of m states by $\mathcal{V} = \{\text{nodes of ST}\}$. A transition from state u to state v is labeled by a if and only if there is in ST an edge from u to v, and v contains the vertex a. We define the Simplex Automaton of K (denoted by $\text{SA}(K)$) as the automaton described above (cf. Figure 4).

SA is basically the same data structure as ST except that the labels are not put on the nodes but on the edges entering these nodes and thus, basic operations in SA can be implemented as in ST. Also, by construction of SA, it is obvious that the number of states and transitions of SA are equal to the number of nodes and edges in ST respectively.

It is known [19] that if a language L is regular (accepted by a DFA) then, L has a unique minimal automaton. DFA minimization is the task of transforming a given DFA into an equivalent DFA that has a minimum number of states. We represent the action of performing DFA minimization by $\mathcal{M}$. The compression of ST can be seen as DFA minimization since merging identical subtrees corresponds to merging indistinguishable states in the automaton. DFA minimization has been well studied. For instance, Hopcroft's algorithm [17] minimizes an automaton with m transitions over an alphabet of size n in $\mathcal{O}(m \log m \log n)$ steps and needs at most $\mathcal{O}(m \log n)$ space. This run-time is shown in [17] to be optimal over the set of regular languages. Additionally, Revuz showed that acyclic automaton (which SA indeed is) can be minimized in linear time [20]. For any $K \in \mathcal{K}_\theta(n, k, d, m)$, let us define the minimal simplex automaton ($\mathcal{M}(\text{SA})$) as the minimal deterministic automaton which recognizes the language L_{K_θ}. In Figure 5, we give the minimal automaton for the complex of Figure 1. Finally, it is possible to get $\mathcal{C}(\text{ST})$ from $\mathcal{M}(\text{SA})$ by duplicating states such that for each node, the labels of all its incoming edges are the same and by moving the labels from the edges to the next node. Now let us look at how to perform operations on $\mathcal{M}(\text{SA})$.

Operations on the Minimal Simplex Automaton

The set of all paths originating from the root are the same in both ST and $\mathcal{M}(\text{SA})$. All operations which involve only traversal along ST are performed with equal (if not better) efficiency in $\mathcal{M}(\text{SA})$ as, for every such operation on ST, we start by traversing from the root. As an example, consider the operation of determining if a simplex σ is in the complex. Let us adapt the algorithm described in [8] to $\mathcal{M}(\text{SA})$. Note that there is a unique path from the initial state which identifies σ in $\mathcal{M}(\text{SA})$. If $\sigma = v_{\ell_0} - \cdots - v_{\ell_{d_\sigma}}$, then from the initial state we go through $d_\sigma + 1$ states by following the transitions $\ell_0, \ldots, \ell_{d_\sigma}$ in that order. If at some point the requisite transition is not found then, declare that the simplex is not in the complex. Hence, performing any static operation i.e. an operation where we don't change $\mathcal{M}(\text{SA})$ in any way, can be carried out in very much the same way in both $\mathcal{M}(\text{SA})$ and ST, although it might be more efficient for $\mathcal{M}(\text{SA})$ as discussed in subsection 3.1 for $\mathcal{C}(\text{ST})$. Addition and deletion of simplices can be trickier in $\mathcal{M}(\text{SA})$ than in ST. We can always expand $\mathcal{M}(\text{SA})$ to SA, (locally) perform the operation and recompress. If the nature of the operation is itself expensive (i.e. worst-case $\Omega(m)$), then the worst-case cost does not change, which is indeed the case for operations such as removal of a face and edge contraction.

We denote by $|\text{SA}|$ and $|\mathcal{M}(\text{SA})|$, the number of states in SA and $\mathcal{M}(\text{SA})$ respectively. Analysis of $|\mathcal{M}(\text{SA})|$ will be done in section 5, after introducing a new data structure in the next section. This is done to put the impact of compression in better perspective.

4 Maximal Simplex Tree

We define the *Maximal Simplex Tree* $\text{MxST}(K)$ as an induced subgraph of $\text{ST}(K)$. All leaves in the ST corresponding to maximal simplices and the nodes encountered on the path from the root to these leaves are kept in the MxST and the remaining nodes are removed. Figure 6 shows the MxST of the simplicial complex given in Figure 1. In $\text{MxST}(K)$, the leaves are in bijection with the maximal simplices of K. Any path starting from the root provides the vertices of a simplex of K. However, in general, not all simplices in K can be associated to a path from the root in $\text{MxST}(K)$.

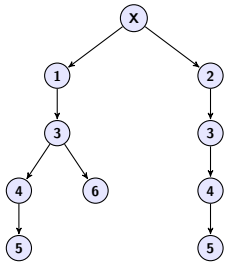

Figure 6 Simplicial Complex of Figure 1 represented using Maximal Simplex Tree.

Table 1 Cost of performing basic operations on MxST.

Operation	Cost
Identifying maximal cofaces of simplex σ / Determining membership of σ	$\mathcal{O}(kd \log n)$
Insertion of a maximal simplex σ	$\mathcal{O}(kd_\sigma \log n)$
Removal of a face	$\mathcal{O}(kd \log n)$
Elementary Collapse	$\mathcal{O}(kd \log n)$
Edge Contraction	$\Theta(kd \log n)$

We note that in MxST we add at most $d+1$ nodes per maximal simplex. Hence, $\text{MxST}(K)$ has at most $k(d+1)+1$ nodes and at most $k(d+1)$ edges (therefore requiring $\mathcal{O}(kd \log n)$ space). We denote by $|\text{MxST}|$ the number of edges in MxST. Since MxST is a factor of ST, the size of MxST is usually much smaller than the size of ST. Further, it always meets the lower bound of Theorem 1, making it a compact data structure. We discuss below the efficiency of MxST in answering queries.

Operations on the Maximal Simplex Tree

In [8] some important basic operations (with appropriate motivation) have been discussed for ST. We will bound now the cost of these operations using MxST. Note that any node in MxST(K) has $\mathcal{O}(n)$ children and we can search for a particular child of a node in time $\mathcal{O}(\log n)$ (using red–black trees). We summarize in Table 1, the asymptotic cost of some basic operations and note that it is already better than ST for some operations. Moreover, we can augment the structure of MxST without paying a lot of extra storage space, so that the above operations can be performed more efficiently. This is explained in section 6.

5 Results on Minimization of the Simplex Automaton

In this section we will see some results, both theoretical and experimental on the minimization of SA, i.e. on the extent of compression of the Simplex Tree.

5.1 Bounds on the Number of States of the Minimal Simplex Automaton

We observe below that the number of leaves in ST is large and grows linearly w.r.t. the number of nodes in ST. The proof follows by a simple induction argument on n.

▶ **Lemma 2.** *If $K \in \mathcal{K}(n,k,d,m)$ then, at least half the nodes of $ST(K)$ are leaves.*

Differently from ST, $\mathcal{M}$(SA) has only one leaf. The following lemma shows that $\mathcal{M}$(SA) has at most half the number of nodes of ST plus one (follows directly from Lemma 2).

▶ **Lemma 3.** *For any $K \in \mathcal{K}(n,k,d,m)$, $\mathcal{M}(SA(K))$ has at most $\frac{m}{2}+1$ states.*

Similar to $\mathcal{M}$(SA), we may define $\mathcal{M}$(MxSA(K)) as the minimal DFA which recognizes only maximal simplices as words. Then the following inequality follows:

▶ **Lemma 4.** *For any pure complex $K \in \mathcal{K}(n,k,d,m)$, $|\mathcal{M}(SA(K))| \geq |\mathcal{M}(MxSA(K))|$.*

Proof Sketch. Notice that every maximal simplex corresponds to a path of exactly $d+1$ transitions and vice versa. Therefore, if all transitions which do not take part in even a single such path are removed, we would obtain the minimized maximal simplex automaton. ◀

In fact, one can prove that for a large class of simplices the equality does not hold. For instance, consider $\mathcal{K}' \subset \mathcal{K}(n,k,d,m)$ such that for any $K \in \mathcal{K}'$ we have that there exists two maximal simplices which have different first letters (i.e. when the simplices are treated as words) but have the same letter at position i, for some i that is not the last position. For this subclass the equality does not hold. Observe also that Lemma 4 holds only for pure simplicial complexes because, if all complexes were allowed, then we will have complexes like in Example 5 where $|\mathcal{M}(\text{SA}(K))| < |\mathcal{M}(\text{MxSA}(K))|$.

▶ **Example 5.** Consider the simplicial complex on 7 vertices given by the following maximal simplices: a 4-cell 1–2–3–6–7, two triangles 2–3–5 and 4–6–7 and an edge 4–5.

5.2 Conditions for Compression

We will first see a result guaranteeing compression regardless of the labeling of the vertices:

▶ **Lemma 6.** *For any pure simplicial complex $K \in \mathcal{K}(n,k,d,m)$, we have that $|\mathcal{M}(SA)|$ is always less than $|SA|$ when $k < d$ and $d \geq 2$.*

Proof Sketch. Since $d > k$ there should exist a free pair (τ, σ) such that τ is a maximal simplex and such that the last two vertices of τ are in σ as well. If s_τ and s_σ are the states recognizing the simplex τ and σ respectively in SA then, they are merged in $\mathcal{M}(\text{SA})$. ◀

In fact, the above result is close to tight: in Example 11 we have a pure simplicial complex with $k = d$ and $|\text{ST}| = |\mathcal{C}(\text{ST})|$. Note that we analyze compression of ST rather than minimization of SA, and we shall continue to do so through out this section because analyzing ST provides better insight into the combinatorial structures which hinder compression.

Intuitively, it seems natural that if the given simplicial complex has a large number of maximal simplices then, regardless of the labeling we should be able to compress some pairs of nodes in MxST. However, Example 7 says otherwise.

▶ **Example 7.** Consider the simplicial complex on $2n$ vertices of dimension $n/2$ defined by the set of maximal simplices given by:

$$\left\{ g(i) \cup \{g^r(i) + n\} \,\middle|\, i \in \left\{1, 2, \ldots, \binom{n}{n/2}\right\}, r \in \{1, 2, \ldots, n/2\} \right\}$$

where g is a bijective map from $\{1, 2, \ldots, \binom{n}{n/2}\}$ to the set of all simplices on n vertices of dimension $n/2 - 1$ and g^r corresponds to picking the r^{th} vertex (in lexicographic order).

Here $k = \frac{n}{2}\binom{n}{n/2} \approx 2^{n-\frac{1}{2}}\sqrt{n/\pi}$ and there is no compression in MxST. Also note that $|\mathcal{C}(\text{MxST})| < |\text{MxST}|$ does not imply $|\mathcal{C}(\text{ST})| < |\text{ST}|$ as can be seen in Example 8.

▶ **Example 8.** Consider the simplicial complex on seven vertices given by the maximal simplices: tetrahedron 1-2-4-6 and three triangles 2-4-5, 3-4-5 and 1-4-7.

In Example 11, we saw a simplicial complex of large dimension which cannot be compressed, but this is due to the way the vertices were labeled. Now, we state a lemma which says that there is always a labeling which ensures compression.

▶ **Lemma 9.** *If $K_\theta \in \mathcal{K}(n, k, d, m)$ with $d > 1$, then we can find a permutation π on $\{1, 2, \ldots, n\}$ such that $|\mathcal{M}(SA(K_{\pi\circ\theta}))| < |SA(K_{\pi\circ\theta})|$.*

We would have liked to obtain better bounds for the size of $\mathcal{M}(\text{SA})$ through conditions just based on n, k, d and m, but sadly this is a hard combinatorial problem. Also, while there is always a good labeling, we show in section 7 that it is NP-Hard to find it.

5.3 Experiments

We define two parameters here, ρ_{ST} and ρ_{MxST}. The first is given by the ratio of $|\text{ST}|$ and $|\mathcal{C}(\text{ST})|$ and the second by the ratio of $|\text{MxST}|$ and $|\mathcal{C}(\text{MxST})|$.

Data Set 1: The set of points were obtained through sampling of a Klein bottle in $\mathbb{R}^5$ and construct the Rips Complex (see [8] for definition) with parameter α using libraries provided by the GUDHI project [16] on input of various values for α. We record in Table 2, $|\mathcal{C}(\text{ST})|$ and $|\mathcal{C}(\text{MxST})|$ for the various complexes constructed.

First, observe that for this set of data, $\frac{\log k}{\log n}$ is small (which is expected for Rips complexes) and thus $|\text{MxST}|$ would be considerably smaller than $|\text{ST}|$. Also, note that MxST hardly compresses but as α increases, ρ_{ST} increases quite fast. This indicates that compression strongly exploits the combinatorial redundancy of ST (i.e. storing each simplex explicitly through a node) in order to compress efficiently.

Table 2 Analysis of experiments on Data Set 1.

No	n	α	d	k	$\lvert\mathrm{ST}\rvert = m - 1$	$\lvert\mathrm{MxST}\rvert$	$\lvert\mathcal{C}(\mathrm{ST})\rvert$	ρ_{ST}	$\lvert\mathcal{C}(\mathrm{MxST})\rvert$	ρ_{MxST}
1	10,000	0.15	10	24,970	604,572	96,104	218,452	2.77	90,716	1.06
2	10,000	0.16	13	25,410	1,387,022	110,976	292,974	4.73	104,810	1.06
3	10,000	0.17	15	27,086	3,543,582	131,777	400,426	8.85	123,154	1.07
4	10,000	0.18	17	27,286	10,508,485	149,310	524,730	20.03	137,962	1.08

Table 3 Analysis of experiments on Data Set 2.

No	n	p	d	k	$\lvert\mathrm{ST}\rvert = m - 1$	$\lvert\mathrm{MxST}\rvert$	$\lvert\mathcal{C}(\mathrm{ST})\rvert$	ρ_{ST}	$\lvert\mathcal{C}(\mathrm{MxST})\rvert$	ρ_{MxST}
1	25	0.8	17	77	315,369	587	467	537.3	121	4.85
2	30	0.75	18	83	4,438,558	869	627	7,079.0	134	6.49
3	35	0.7	17	181	3,841,590	1,592	779	4,931.4	245	6.50
4	40	0.6	19	204	9,471,219	1,940	896	10,570.6	276	7.03
5	50	0.5	20	306	25,784,503	2,628	1,163	22,170.7	397	6.62

Data Set 2: All experiments conducted above are for Rips complexes with $\frac{d}{n}$ small. We now check the extent of compression for simplicial complexes with large $\frac{d}{n}$. To the aim, we look at flag complexes generated using a random graph $G_{n,p}$ on n vertices where a pair of vertices share an edge with probability p, and record in Table 3, $\lvert\mathcal{C}(\mathrm{ST})\rvert$ and $\lvert\mathcal{C}(\mathrm{MxST})\rvert$ for the various complexes constructed.

Here we observe staggering values for ρ_{ST} which only seems to grow as larger simplicial complexes were constructed. This is primarily because random simplicial complexes don't behave like pathological simplicial complexes which hinder compression; it is rare that there exists both large cliques and a large fraction of low dimensional maximal simplices.

6 Simplex Array List

In this section, we build a new data structure which is a hybrid of ST and MxST. The *Simplex Array List* $\mathrm{SAL}(K)$ is a (rooted) directed acyclic graph on at most $k\left(\frac{d(d+1)}{2}+1\right)$ nodes with maximum out-degree d, which is obtained by modifying MxST or constructed from the maximal simplices of K. Intuitively, SAL is representing K by storing all the edges of K explicitly as nodes in $\mathrm{SAL}(K)$ and the edges in $\mathrm{SAL}(K)$ are used to capture the incidence relations between simplices. More precisely, a path of length j in $\mathrm{SAL}(K)$ corresponds to a unique j-simplex in K. We describe the construction of SAL below.

6.1 Construction

We will first see how to obtain SAL from MxST by performing three operations which we define below.

1. **Unprefixing** ($\mathcal{U}$)**:** Excluding the root and the leaves, for every node v in MxST with outdegree d_v, duplicate it into d_v nodes with outdegree 1, (one copy of v for each of its children) by starting from the parents of the leaves and recursively moving up in the tree.
2. **Transitive Closure** ($\mathcal{T}$)**:** For every pair of nodes (u, v) in $\mathcal{U}(\mathrm{MxST})$ (u not being the root), if there is a path from u to v, then add an edge from u to v in $\mathcal{T}(\mathcal{U}(\mathrm{MxST}))$ (if it doesn't already exist).
3. **Expanding Representation** ($\mathcal{R}$)**:** For every node v in $\mathcal{T}(\mathcal{U}(\mathrm{MxST}))$ with outdegree d_v, duplicate it into d_v nodes with outdegree 1, i.e. one copy of v for each of its children, by

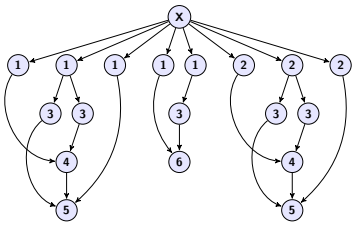

Figure 7 Simplicial Complex of Figure 1 represented using $\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST})))$.

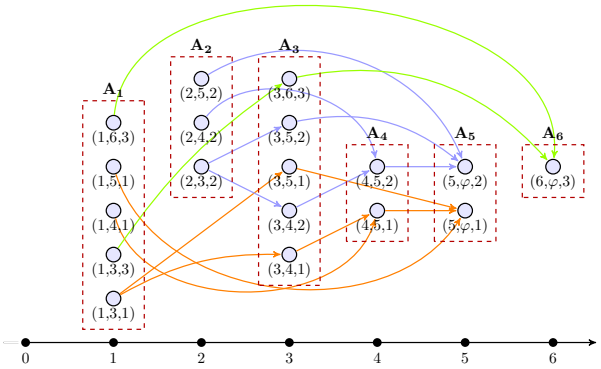

Figure 8 Simplex Array List for complex in Figure 1 embedded on the number line.

starting from the children of the root and recursively moving down to children of smallest label. Therefore, if we append i empty labels at the end of each maximal simplex, $\mathcal{R}$ applied i times will give a graph where every node uniquely represents a i-simplex.

SAL can be seen as $\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST})))$ and each node uniquely represents an edge in the complex. Figure 7 shows $\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST})))$ of the simplicial complex given in Figure 1.

We will now see an equivalent construction of SAL from its maximal simplices and it is this construction we will use to perform operations. For a given maximal simplex $\sigma = v_{\ell_0} \cdots v_{\ell_j}$, associate a unique key between 1 and k generated using a hash function $\mathcal{H}$ and then introduce $\frac{j(j+1)}{2} + 1$ new nodes in SAL. We build a set of $\frac{j(j+1)}{2} + 1$ labels and assign uniquely a label to each node. The set of labels is defined as the union of the following two sets: $S_1 = \{(\ell_i, \ell_{i'}, \mathcal{H}(\sigma)) \mid i \in \{0, 1, \ldots, j-1\}, i' \in \{i+1, \ldots, j\}\}$ and $S_2 = \{(\ell_j, \varphi, \mathcal{H}(\sigma))\}$, where φ denotes an empty label (cf. Figure 8 for an example). We introduce an edge from node with label $(\ell_p, \ell_{p'}, \mathcal{H}(\sigma))$ to node with label $(\ell_q, \ell_{q'}, \mathcal{H}(\sigma))$ if and only if $p' = q$. Additionally, we introduce an edge from every node with label $(\ell_p, \ell_j, \mathcal{H}(\sigma))$ in S_1 to the node with label $(\ell_j, \varphi, \mathcal{H}(\sigma))$ in S_2. Thus, in SAL we represent a maximal j-simplex using a connected component containing $|S_1| + |S_2| = \frac{j(j+1)}{2} + 1$ nodes and $\frac{j(j^2+5)}{6}$ directed edges. To perform basic operations efficiently, we embed SAL on the number line such that for every $i \in \{1, 2, \ldots, n\}$ on the number line we have an array A_i of nodes which has labels of the form (i, i', z) for some $z \in \{1, \ldots, k\}$ and $i' \in \{i+1, \ldots, n, \varphi\}$. Sort each A_i based on i' and in case of ties, sort them based on z.

If the root is removed in $\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST})))$, the graph we get is the same as the one described in the previous paragraph. Labels (as described above) for the nodes in $\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST})))$ can be easily given by just looking at the vertex represented by the node, and its children. Further, the number of nodes and number of edges in SAL are both invariant over the labeling of the vertices because SAL is constructed from $\mathcal{U}(\text{MxST})$.

6.2 Some Observations about the Simplex Array List

SAL(K) has at most $k\left(\frac{d(d+1)}{2} + 1\right)$ nodes. Also, for each maximal simplex of dimension d_σ, the outdegree of any node in the connected component corresponding to the maximal simplex, is at most d_σ. Therefore, the total number of edges in SAL(K) is at most $k\left(\frac{d^2(d+1)}{2} + d\right)$.

Hence, the space required to store SAL(K) is $\mathcal{O}(kd^3 \log n)$. Also, unless otherwise stated |SAL| refers to number of edges in SAL.

We now see that, differently from MxST, the simplices of K are all associated with paths in SAL(K). We say a path p is associated to a simplex σ if the sequence of numbers obtained by looking at the corresponding nodes which are embedded on the number line along p are exactly the labels of the vertices of σ in lexicographic order.

▶ **Lemma 10.** *Any path in SAL(K) is associated to a simplex of K and any simplex of K is associated to at least one such path.*

Observe that several paths can provide the same simplex since a simplex may appear in several maximal simplices. Hence, the vertices of a given simplex cannot be accessed in a deterministic way. The previous lemma together with this observation implies that SAL is a non-deterministic finite automaton (NFA). NFA are a natural generalization of DFA. The size of a NFA is smaller than that of a DFA detecting the same language, but the operations on NFA take in general more time. We demonstrate the above fact using Example 11.

▶ **Example 11.** Let $K \in \mathcal{K}(2k+1, k, k, m)$ be defined on the vertices $\{1, \ldots, 2k+1\}$ and the set of maximal simplices be given by $\{(\{1, \ldots, k+1\} \setminus \{i\}) \cup \{k+1+i\} \mid 1 \leq i \leq k\}$.

Thus SAL(K) has $\frac{k^2(k+1)}{2} + k$ nodes while $\mathcal{M}(\text{SA}(K))$ has at least 2^k states (all states reached after reading the words $s \subseteq \{1, \ldots, k\}$ are pairwise distinct). Moreover, this motivates the need for considering SAL over $\mathcal{M}$(SA), as the gap in their sizes can be exponential.

Building SAL(K) can be seen as partially compressing the ST(σ) associated to each maximal simplex σ (where σ and its subfaces are seen as a subcomplex). Compressing ST(σ) will lead to a subtree which is exactly the same as the transitive closure of MxST(σ). Therefore, collecting all $\mathcal{C}(\text{ST})(\sigma)$ for all maximal simplices σ and merging the roots is the same as $\mathcal{T}(\mathcal{U}(\text{MxST}(K)))$. Now applying $\mathcal{R}$ on $\mathcal{T}(\mathcal{U}(\text{MxST}(K)))$ can be seen as an act of uncompression. We apply $\mathcal{R}$ once to ensure that for every node, all its children represent the same vertex and thus belong to the same A_i. If $\mathcal{R}$ is applied multiple times then, it is equivalent to duplicating nodes (seen as an act of uncompression) to get all children of a node closer together inside A_i. Next, we discuss below how to perform operations in SAL at least as efficiently as in ST.

6.3 Operations on the Simplex Array List

Let us now analyze the cost of performing basic operations on SAL (the motivation behind these operations are well described in [8]). Denote by $\Gamma_j(\sigma, \tau)$ the number of maximal simplices that contain a j-simplex τ which is in σ. Define $\Gamma_j(\sigma) = \max_{\tau} \Gamma_j(\sigma, \tau)$ and $\Gamma_j = \max_{\sigma \in K} \Gamma_j(\sigma)$. It is easy to see that $k \geq \Gamma_0 \geq \Gamma_1 \geq \cdots \geq \Gamma_d = 1$. In the case of SAL, we are interested in the value of Γ_1 which we use to estimate the worst-case cost of basic operations in SAL.

Membership of Simplex. To determine membership of $\sigma = v_{\ell_0} \cdots v_{\ell_{d_\sigma}}$ in K, first determine the contiguous subarray of $A_{\ell_0}, \ldots, A_{\ell_{d_\sigma}}$, say $B_{\ell_0}, \ldots, B_{\ell_{d_\sigma}}$ such that every B_{ℓ_i} contains all nodes with labels of the form (ℓ_i, ℓ_{i+1}, z), for some z (B_{ℓ_i}'s indeed form a contiguous subarray because of the way elements in A_{ℓ_i} were sorted). We emphasize here that we determine each B_{ℓ_i} only by its starting and ending location in A_{ℓ_i} and do not explicitly read the contents of each element in B_{ℓ_i}. Thus, if P is a projection function such that $P((\ell_i, \ell_{i+1}, z)) = z$ then, we see each $P(B_{\ell_i})$ as a subset of $\{1, \ldots, k\}$ because the only part of the label that distinguishes

two elements in B_{ℓ_i} is the hash value of the maximal simplex. Now we have $\sigma \in K$ if and only if $\bigcap_{0 \le i \le d_\sigma} P(B_{\ell_i}) \neq \emptyset$. This is because if $\sigma \in K$ then, from Lemma 10 there should exist a path corresponding to this simplex which would imply $\bigcap_i P(B_{\ell_i}) \neq \emptyset$, and if $m \in \bigcap_i P(B_{\ell_i})$, then σ is a face of m. Computing the intersection can be done in $\mathcal{O}(\gamma d_\sigma \log \zeta)$ time, where $\gamma = \min_i |B_{\ell_i}|$ and $\zeta = \max_i |A_{\ell_i}|$. Computing the subarrays can be done in $\mathcal{O}(d_\sigma \log \zeta)$ time. Thus total running time is $\mathcal{O}(d_\sigma(\gamma \log \zeta + \log \zeta)) = \mathcal{O}(d_\sigma \Gamma_1 \log(kd))$.

For example, consider the SAL of figure 8 and the task of checking membership of $\sigma = 2-3-5$ in the complex of figure 1. Then, we have $B_2 = \{(2,3,2)\}, B_3 = \{(3,5,1),(3,5,2)\}$, and $B_5 = \{(5,\varphi,1),(5,\varphi,2)\}$. We see each $P(B_i)$ as a subset of $\{1,2,3\}$ as follows: $P(B_2) = \{2\}, P(B_3) = \{1,2\}$, and $P(B_5) = \{1,2\}$. Clearly $\bigcap_i P(B_i) = \{2\}$ and σ is indeed a face of the second maximal simplex $2-3-4-5$.

Insertion. Suppose we want to insert a maximal simplex σ then, building a connected component takes time $\mathcal{O}(d_\sigma^3)$. Updating the arrays A_i takes time $\mathcal{O}(d_\sigma^2 \log \zeta)$. Next, we have to check if there exist maximal simplices in K which are now faces of σ, and remove them. We consider every edge σ_Δ in σ and compute Z_Δ the set of all maximal simplices which contain σ_Δ (which can be done in time $\mathcal{O}(d_\sigma^3 \Gamma_1 \log(kd))$). Then, we compute $\bigcup_{\sigma_\Delta \in \sigma} Z_\Delta$ whose size is at most $d_\sigma^2 \Gamma_1$ and check if any of these maximal simplices are faces in σ (can be done in $\mathcal{O}(d_\sigma^3 \Gamma_1)$ time). To remove all such faces of σ which were previously maximal takes at the most $\mathcal{O}(d_\sigma^4 \Gamma_1)$ time. Therefore, total time for insertion is $\mathcal{O}(d_\sigma^3 \Gamma_1 (d_\sigma + \log(kd)))$.

Removal. To remove a face σ, obtain the maximal simplices which contain it (can be done in $\mathcal{O}(d_\sigma \Gamma_1 \log(kd))$ time) and for each of them make d_σ copies of the connected component, and in the i^{th} copy delete all nodes with label (σ_i, x, y) for some x, y, and where σ_i denotes the label of the i^{th} vertex of σ. Thus, the total running time is $\mathcal{O}(d_\sigma d^3 \Gamma_1 \log(kd))$.

Elementary Collapse. Given a pair of simplices (σ, τ), first check if it is a free pair. This is done by obtaining a list of all maximal simplices which contain σ, through a membership query (costs $\mathcal{O}(d_\sigma \Gamma_1 \log(kd))$ time) and then checking if τ is the only member in that list. If yes, remove σ (and add its facets). This takes time $\mathcal{O}(d_\sigma^4)$. Thus, total running time is $\mathcal{O}(d_\sigma(d_\sigma^3 + \Gamma_1 \log(kd)))$.

Edge Contraction. Here we cannot do better than rebuilding the entire SAL (as in MxST) and therefore the cost of the operation is $\mathcal{O}(kd^3)$. However, it's not really bad as size of SAL is already smaller than the size of ST which takes time proportional to $\mathcal{O}(md + k2^d \log n)$ to perform edge contraction.

We summarize in Table 4 the asymptotic cost of the basic operations discussed above and compare it with ST and MxST, through which the efficiency of SAL is established.

Filtration. We know from Lemma 10 that to every simplex in the complex, we can associate a set of paths in SAL. This can be used to store filtration in some cases. However, if a data structure needs to support all possible filtrations then, it can be provably shown that there

* We would like to recapitulate here the lower bound from Theorem 1 of $\Omega(kd \log n)$.
† The space needed to represent ST is $\Theta(m \log n)$ which is written as $\mathcal{O}(k2^d \log n)$ to help in comparison.

Table 4 Cost of performing basic operations on SAL in comparison with ST and MxST.

	ST	MxST	SAL
Storage*	$\mathcal{O}(k2^d \log n)^\dagger$	$\mathcal{O}(kd \log n)$	$\mathcal{O}(kd^3(\log n + \log k))$
Membership of a simplex σ	$\mathcal{O}(d_\sigma \log n)$	$\mathcal{O}(kd \log n)$	$\mathcal{O}(d_\sigma \Gamma_1 \log(kd))$
Insertion of a maximal simplex σ	$\mathcal{O}(2^{d_\sigma} d_\sigma \log n)$	$\mathcal{O}(kd_\sigma \log n)$	$\mathcal{O}(d_\sigma^3 \Gamma_1 (d_\sigma + \log(kd)))$
Removal of a face	$\mathcal{O}(m \log n)$	$\mathcal{O}(kd \log n)$	$\mathcal{O}(d_\sigma d^3 \Gamma_1 \log(kd))$
Elementary Collapse	$\mathcal{O}(2^{d_\sigma} \log n)$	$\mathcal{O}(kd \log n)$	$\mathcal{O}(d_\sigma (d_\sigma^3 + \Gamma_1 \log(kd^2)))$
Edge Contraction	$\mathcal{O}(k2^d \log n)$	$\mathcal{O}(kd \log n)$	$\mathcal{O}(kd^3)$

Table 5 Values of Γ_0, Γ_1, Γ_2, and Γ_3 for the simplicial complexes generated from Data Set 1.

No	n	α	d	k	m	Γ_0	Γ_1	Γ_2	Γ_3	|SAL|
1	10,000	0.15	10	24,970	604,573	62	53	47	37	424,440
2	10,000	0.16	13	25,410	1,387,023	71	61	55	48	623,238
3	10,000	0.17	15	27,086	3,543,583	90	67	61	51	968,766
4	10,000	0.18	17	27,286	10,508,486	115	91	68	54	1,412,310

is no better way to do so, than by storing the filtration value explicitly for each simplex in the complex. Therefore one cannot hope to find a more compact representation than ST in order to support all possible filtrations.

Performance of SAL. Plainly, if the number of maximal simplices is small (i.e. can be treated as a constant), SAL and MxST are very efficient data structures and this is indeed the case for a large class of complexes encountered in practice as discussed in section 2.

Remarkably, even if k is not small but d is small then, SAL is a compact data structure as given by the lower bound in Theorem 1. This is because $\mathcal{O}(kd^3(\log n + \log k))$ bits are sufficient to represent SAL and the lower bound is met when d is fixed (as it translates to needing $\mathcal{O}(k \log n)$ bits to represent SAL). Also, it is worth noting here that Γ_0 is usually a small fraction of k and since Γ_1 is at most Γ_0, the above operations are performed considerably faster than in MxST where almost always the only way to perform operations is to traverse the entire tree. Indeed SAL was intended to be efficient in this regard as even if k is not small the construction of SAL replaces the dependence on k by a dependence on a more local parameter Γ_1 that reflects some "local complexity" of the simplicial complex. As a simple demonstration, we estimated $\Gamma_0, \Gamma_1, \Gamma_2$, and Γ_3 for the simplicial complexes of Data Set 1 (see section 5.3). These values are recorded in Table 5.

It is interesting to note that |SAL| is larger than $|\mathcal{C}(\text{ST})|$ but much smaller than |ST|. This is expected, as SAL promises to perform basic operations more efficiently than ST while compromising slightly on size. Further our intuition as described previously was that Γ_0 should be much smaller than k, and this is supported by the above results. Also, we note that for larger simplicial complexes such as complexes No 3 and 4, there is a significant gap between Γ_0 and Γ_1. Since complexity of basic operations using SAL is parametrized by Γ_1 (and not Γ_0), the above results support our claim that SAL is an efficient data structure.

Local Sensitivity of Simplex Array List. We note here that while the cost of basic operations are bounded using Γ_1, we could use local parameters such as γ and Z_Δ (see previous paragraphs on Membership of Simplex and Insertion for definition) to get a better estimate

on the cost of these operations. γ captures local information about a simplex σ sharing an edge with other maximal simplices of the complex. More precisely, it is the minimum, over all edges of σ, of the largest number of maximal simplices that contain the edge. If σ has an edge which is contained in a few maximal simplices then, γ is very small. Z_Δ captures another local property of a simplex σ – the set of all maximal simplices that contain the edge σ_Δ. Therefore, SAL is indeed sensitive to the local structure of the complex.

7 Labeling Dependency

In this section, we discuss how the labeling of the vertices affects the size of the data structures discussed in this paper. In particular, both MxST and $\mathcal{M}$(SA) are not label invariant like ST and SAL. To see this, consider a simplicial complex which contains a maximal triangle and a maximal tetrahedron, sharing an edge. We could label the triangle and tetrahedron as 1–2–3 and 1–2–4–5, or as 1–3–4 and 2–3–4–5 respectively. Note that the two labelings give two $\mathcal{M}$(SA) (and MxST) of different sizes. We skip all the proofs in this section and instead direct the reader to the full version of the paper [7].

First, we formalize the label ordering problem on MxST (and $\mathcal{C}$(MxST), $\mathcal{C}$(ST), and $\mathcal{M}$(SA)) as follows: Given an integer α and a simplicial complex $K_\theta \in \mathcal{K}_\theta(n,k,d,m)$, does there exist a permutation π of $1,2,\ldots,n$ such that $|\text{MxST}(K_{\pi\circ\theta})| \leq \alpha$ (similarly we ask $|\mathcal{C}(\text{MxST}(K_{\pi\circ\theta}))| \leq \alpha$, $|\mathcal{C}(\text{ST}(K_{\pi\circ\theta}))| \leq \alpha$, and $|\mathcal{M}(\text{SA}(K_{\pi\circ\theta}))| \leq \alpha$)? Let us refer to this problem as MxSTMINIMIZATION(K_θ,α) (similarly we have CMxSTMINIMIZATION(K_θ,α), CSTMINIMIZATION(K_θ,α), and MSAMINIMIZATION(K_θ,α)). We have the following results:

▶ **Theorem 12** ([9]). *MxSTMINIMIZATION is NP-Complete.*

▶ **Theorem 13.** *CMxSTMINIMIZATION, CSTMINIMIZATION, and MSAMINIMIZATION are all NP-Complete.*

8 Discussion and Conclusion

In this paper, we introduced a compression technique for the Simplex Tree without compromising on functionality. Additionally, we have proposed two new data structures for simplicial complexes – the Maximal Simplex Tree and the Simplex Array List. We observed that the Minimal Simplex Automaton is generally smaller than the Simplex Automaton. Further, we showed that the Maximal Simplex Tree is compact and that the Simplex Array List is efficient (and compact when d is fixed). This is summarized in Table 4.

The transitive closure of MxST may have a node, with as many as kd outgoing edges to neighbors containing the same label. SAL reduces the number of outgoing edges to such neighbors with the same label from kd to d, making it much more powerful. In short, it reduces the non-determinism of their equivalent automaton representation. Also, most complexes observed in practice have k to be a low degree polynomial in n. Example 11 and Lemma 6 both deal with complexes where k is small. Further, all hardness results in section 7 are for complexes of dimension at most 2. Thus, complexes where either k or d is small are interesting to study and for these cases, SAL is very efficient.

Trie Compression, like that of $\mathcal{M}$(SA), are efficient techniques when the trie is assumed to be static. However, over the last decade, this has been extended using Dynamic Minimization – the process of maintaining an automaton minimal when insertions or deletions are performed. This has been well studied in [21], and extended to acyclic automata in [12] which would be of particular interest to us.

Another direction, is to look at approximate data structures for simplicial complexes, i.e. we store almost all the simplices (introducing an error) and gain efficiency in compression (i.e. little storage). This is a well explored topic in automata theory called hyperminimization [18] and since our language is finite, $k-$minimization [5] and cover automata [10] might give efficient approximate data structures by hyperminimizing SA.

Theorem 13 provides a new dimension to the hardness results obtained by Comer and Sethi in [11]. It would be worth exploring this direction further. Also, it would be interesting to find approximation algorithms for MSAMINIMIZATION. Finally proving better bounds on extent of compression remains an open problem and may be geometric constraints will eliminate pathological examples which hinder in proving good bounds on compression.

Acknowledgement. We would like to thank Eylon Yogev for helping with carrying out some experiments.

References

1 A. Acharya, H. Zhu, and K. Shen: Adaptive Algorithms for Cache-efficient Trie Search, *In Workshop on Algorithm Engineering and Experimentation ALENEX 99, Baltimore*, 1999.

2 A. Andersson and S. Nilsson: Improved Behaviour of Tries by Adaptive Branching, *In Information Processing Letters*, Vol 46, pages 295–300, 1993.

3 A. W. Appel and G. J. Jacobson: The world's fastest scrabble program, *In Communications of the ACM*, Vol 31, 1988.

4 D. Attali, A. Lieutier, and D. Salinas: Efficient data structure for representing and simplifying simplicial complexes in high dimensions. *In International Journal of Computational Geometry and Applications, 22(4)*, pages 279–303, 2012.

5 A. Badr, V. Geffert, and I. Shipman: Hyper-minimizing minimized deterministic finite state automata. *In RAIRO Theoretical Informatics and Applications*, pages 69–94, 2009.

6 L. J. Billera and A. Björner: Face numbers of polytopes on complexes. *In Handbook of Discrete and Computational Geometry*, CRC Press, pages 291–310, 1997.

7 J.-D. Boissonnat, Karthik C. S., and S. Tavenas: Building efficient and compact data structures for simplicial complexes. `http://arxiv.org/abs/1503.07444`.

8 J.-D. Boissonnat and C. Maria: The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes. *In Algorithmica 70(3)*, pages 406–427, 2014.

9 J.-D. Boissonnat and D. Mazauric: On the complexity of the representation of simplicial complexes by trees. `http://hal.inria.fr/hal-01089846`

10 C. Câmpeanu, N. Sântean, and S. Yu: Minimal cover-automata for finite languages. *In Theoretical Computer Science* 267(1–2), pages 3–16, 2001.

11 D. Comer and R. Sethi: Complexity of Trie Index Construction, *In Proceedings of Foundations of Computer Science*, pages 197-207, 1976.

12 J. Daciuk, S. Mihov, B. Watson, and R. Watson: Incremental construction of minimal acyclic finite-state automata. *In Comput. Linguist.*, Volume 26, pages 3–16, 2000.

13 D. Eppstein, M. Löffler, and D. Strash: Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time, *In ISAAC (1)*, pages 403–414, 2010.

14 M. Golumbic: Algorithmic Graph Theory and Perfect Graphs. *In Academic Press*, 2004.

15 M. Grohe, S. Kreutzer, and S. Siebertz: Characterisations of Nowhere Dense Graphs, *In FSTTCS 13*, pages 21–40, 2013.

16 GUDHI – Geometric Understanding in Higher Dimenions. `https://project.inria.fr/gudhi/`

17 J. Hopcroft: An n log n algorithm for minimizing states in a finite automaton. *In Theory of machines and computations*, pages 189–196, 1971.

18 A. Maletti: Notes on hyper-minimization. *In Proceedings 13th International Conference Automata and Formal Languages*, pages 34–49, 2011.

19 A. Nerode: Linear Automaton Transformations, *In Proceedings of the American Mathematical Society*, Volume 9, pages 541–544, 1958.

20 D. Revuz: Minimisation of acyclic deterministic automata in linear time, *In Theoretical Computer Science, Volume 92, Issue 1*, pages 181–189, 1992.

21 K. Sgarbas, N. Fakotakis, and G. Kokkinakis: Optimal insertion in deterministic DAWGs. *In Theoretical Computer Science*, pages 103–117, 2003.