Building Efficient and Compact Data Structures for Simplicial Complexe

Jean-Daniel Boissonnat, Karthik C. Srikanta, Sébastien Tavenas

Algorithmica manuscript No.
(will be inserted by the editor)

Building Efficient and Compact Data Structures for Simplicial Complexes

Jean-Daniel Boissonnat · Karthik C. S. · Sébastien Tavenas

Received: date / Accepted: date

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 the Maximal Simplex Tree (MxST) and the Simplex Array List (SAL). We analyze the compressed Simplex Tree, the Maximal Simplex Tree, and the Simplex Array List under various settings.

Keywords Simplicial complex · Compact data structures · Automaton · NP-hard.

An extended abstract appeared in the proceedings of the 31st International Symposium on Computational Geometry.

Jean-Daniel Boissonnat
Geometrica, INRIA Sophia Antipolis - Méditerranée, France.
E-mail: Jean-Daniel.Boissonnat@inria.fr.
This work was partially supported by the Advanced Grant of the European Research Council GUDHI (Geometric Understanding in Higher Dimensions).

Karthik C. S.
Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Israel.
E-mail: karthik.srikanta@weizmann.ac.il.
This work was partially supported by Irit Dinur's ERC-StG grant number 239985. Some parts of this work were done at ENS Lyon and at University of Nice - Sophia Antipolis, and were supported by LIP fellowship and Labex UCN@Sophia scholarship respectively.

Sébastien Tavenas
Microsoft Research India.
E-mail: t-sebat@microsoft.com.
A part of this work was done at LIP, ENS Lyon (UMR 5668 ENS Lyon - CNRS - UCBL - INRIA, Université de Lyon).

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. [2] proposed the skeleton-blockers data structure which represent 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 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 does not 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 efficiently). Moreover, the tree structure of the Simplex Tree leads to efficient implementation of the 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 [3] 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 [4] where trie data structures are optimized for computer memory architectures. Other popular compact representations for tries in connection with predictive text compression are discussed in [27], but they only include (all) substrings of constant length that exist in a text and also do not focus on supporting efficient access in the compressed trie. Therefore, such representations are not useful here, due to the loss of significant information.

When the size of the structure is of primary concern, the focus is usually 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 [1]. 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.

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 this data structure is compact and that it allows efficient operations. MxST is 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 simplicial complex is fixed, a case of great interest in Manifold Learning and Topological Data Analysis.

2 Simplicial Complex: Definitions and a Lower Bound

A simplicial complex K is defined over a (finite) vertex set V whose elements are called the vertices of K and is a set of non-empty subsets of V that is required to satisfy the following two conditions:

1. $p \in V \Rightarrow \{p\} \in K$
2. $\sigma \in K, \tau \subseteq \sigma \Rightarrow \tau \in K$

Each element $\sigma \in K$ is called a simplex or a face of K and, if $\sigma \in K$ 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 K is the largest d such that it contains a d-simplex.

A face of a simplex $\sigma = \{p_0, ..., p_s\}$ is a simplex whose vertices form a subset of $\{p_0, ..., p_s\}$. A proper face is a face different from σ and the facets of σ are its proper faces of maximal dimension. A simplex $\tau \in K$ admitting σ as a face is called a coface of σ.

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 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.

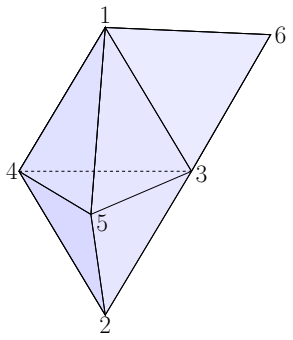

Fig. 1: Simplicial complex with the tetrahedra 1–3–4–5 and 2–3–4–5, and the triangle 1–3–6.

The flag complex of an undirected graph G is defined as an abstract simplicial complex, whose simplices are the sets of vertices in the cliques of G. Let (P, d) be a metric space where P is a discrete point set. Given a positive real number $r > 0$, the Rips complex is the abstract simplicial complex $\mathcal{R}^r(P)$ where a simplex $\sigma \in \mathcal{R}^r(P)$ if and only if $d(p, q) \leq r$ for every pair of vertices of σ. Note that the Rips complex is a special case of a flag complex. This completes the definition of the complexes which will be used in this 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 [15], and in general, from nowhere dense graphs [18], and also from chordal graphs [16]. 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 $k = n^{\mathcal{O}(1)}$ [15]. 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 class of all simplicial complexes on n vertices of dimension d, containing k maximal simplices, 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 The proof of the first statement is by contradiction. Let us define $h = k - n \geq 1$ and suppose that there exists a data structure that can be stored using only $s < \log \alpha \stackrel{\text{def}}{=} \log \binom{\binom{n/2}{d+1}}{h}$ 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. By the pigeon hole principle, two different simplicial complexes, say K and K', are encoded by the same word. So any algorithm will give the same answer for K and K'. But, by the construction of these complexes, there is a simplex which is in K and not in K'. This leads to a contradiction.

The simplicial complexes are constructed as follows. Let $P' \subset P$ be a subset of cardinality $n/2$, and consider the set of all possible simplicial complexes of dimension d with vertices in P' that contain h maximal simplices. We further assume that all maximal simplices have dimension d exactly. These complexes are $\alpha = \binom{\binom{n/2}{d+1}}{h}$ in number, since the total number of maximal d dimensional simplices is $\binom{n/2}{d+1}$ and we choose h of them. Let us call them $\Gamma_1, \ldots, \Gamma_\alpha$. We now extend each Γ_i so as to obtain a simplicial complex whose vertex set is P and has exactly k maximal simplices. The maximal simplices will consist of the h maximal simplices of dimension d already constructed plus a number of maximal simplices of dimension 1. The set of vertices of Γ_i, $\mathrm{vert}(\Gamma_i)$, may be a strict subset of P'. Let its cardinality be $\frac{n}{2} - r_i$ and observe that $0 \leq r_i < \frac{n}{2}$. Consider now the complete graph on the $\frac{n}{2} + r_i$ vertices of $P \setminus \mathrm{vert}(\Gamma_i)$. Any spanning tree of this graph gives $\frac{n}{2} + r_i - 1$ edges and we arbitrarily choose $\frac{n}{2} - r_i + 1$ edges from the remaining edges of the graph to obtain n distinct edges spanning over the vertices of $P \setminus \mathrm{vert}(\Gamma_i)$. We have thus constructed a 1–dimensional simplicial complex K_i on the $\frac{n}{2} + r_i$ vertices of $P \setminus \mathrm{vert}(\Gamma_i)$ with exactly n maximal simplices. Finally, we define the complex $\Lambda_i = \Gamma_i \cup K_i$ that has P as its vertex set, dimension d, and k maximal simplices. The set of Λ_i, $i = 1, \cdots, \alpha$, is the set of simplicial complexes we were looking for.

The second statement in the theorem is proved through the following computation:

$$
\begin{aligned}
\log \binom{\binom{n/2}{d+1}}{k-n} &\geq \log \left(\frac{n^{(d+1)(k-n)}}{2^{(d+1)(k-n)}(d+1)^{(d+1)(k-n)}(k-n)^{(k-n)}} \right) \\
&= (d+1)(k-n)\log n - (d+1)(k-n) \\
&\quad - (d+1)(k-n)\log(d+1) - (k-n)\log(k-n) \\
&> (d+1)(k-n)\log n - 3(d+1)(k-n) \\
&\quad - (d+1)(k-n)\log d - (k-n)\log k \\
&\geq (d+1)(k-n)(\log n - 3 - \log d) - (k-n)(1-\varepsilon)d\log n \\
&\geq d\varepsilon(k-n)\log n + (k-n)\log n - (d+1)(k-n)(3 + \frac{\varepsilon}{3}\log n) \\
&\geq \frac{2\varepsilon}{3}\left(1 - \frac{\varepsilon}{2}\right) kd\log n + (k-n)\log n - 3d(k-n) \\
&\quad - (k-n)(3 + \frac{\varepsilon}{3}\log n)
\end{aligned}
$$

$$= \Omega(kd \log n)$$

We note that in the above computation, the first inequality is obtained by applying the following bound on binomial coefficients: $\binom{n}{d} \geq \left(\frac{n}{d}\right)^d$. □

We can adapt the above proof to build n maximal simplices on $P \setminus \text{vert}(\Gamma_i)$ each of dimension d, to ensure the lower bound applies also to pure simplicial complexes. This is done by first building $\lfloor \frac{|P \setminus \text{vert}(\Gamma_i)|}{d+1} \rfloor$ disjoint maximal simplices of dimension d on vertices of $P \setminus \text{vert}(\Gamma_i)$ and then, building one maximal simplex which contains all the remaining vertices. We would complete the construction of k maximal simplices in the complex by choosing $n - \lfloor \frac{|P \setminus \text{vert}(\Gamma_i)|}{d+1} \rfloor - 1$ new maximal simplices of dimension d from vertices of $P \setminus \text{vert}(\Gamma_i)$.

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) [5]. 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 last label of the word representation of a simplex σ will be called the last label of σ and denoted by $\text{last}(\sigma)$. 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, we store the corresponding words in a tree satisfying the following properties:

1. The nodes of the 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, the node N associated to a simplex $\sigma \neq \emptyset$ stores the label of the vertex $\text{last}(\sigma)$.
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.

This data structure is called the Simplex Tree of K [8] and denoted by $\text{ST}(K)$ or simply ST when there is no ambiguity. It may be seen as a trie [10] 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. Also, in this paper we assume that ST is directed from the root to the leaves.

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 Simplex Tree. 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.

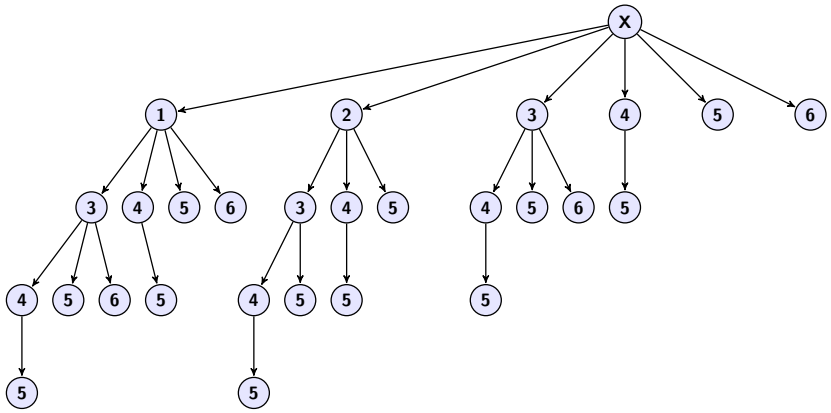

Fig. 2: Simplex Tree of the simplicial complex in Figure 1.

If K consists of m simplices (including the empty face), 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 the Simplex Tree 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 introduce a compression technique for the ST.

3.1 Compressed Simplex Tree

Consider the ST in Figure 3 and note that the red shaded region appears 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 4, 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.**

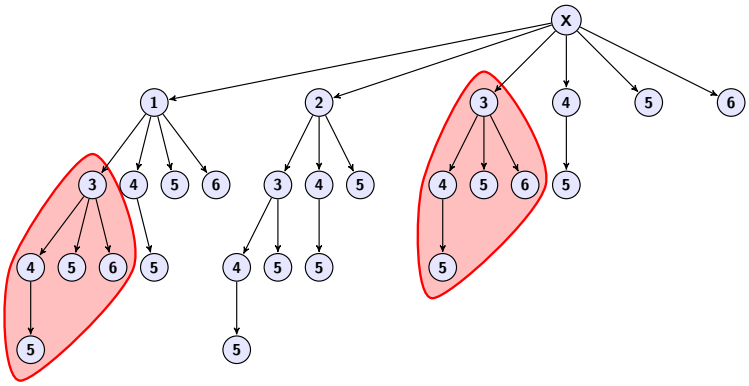

Fig. 3: Common subtrees of the Simplex Tree in Figure 2.

Answering simplex membership queries and other queries that only require traversing 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. However, in $\mathcal{C}(\mathrm{ST})$, parents are not unique. To account for this, we mark the parents that 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 ST. We will also describe insertion and removal operations on $\mathcal{C}(\mathrm{ST})$ 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:

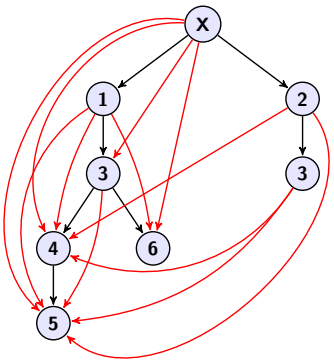

Fig. 4: Compressed Simplex Tree of the Simplex Tree given in Figure 2. The extent of compression is demonstrated by the following: the edge $4-5$ which appears six times in the Simplex Tree of Figure 2, appears only once in the Compressed Simplex Tree

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 SA(K)) as the automaton described above (cf. Figure 5).

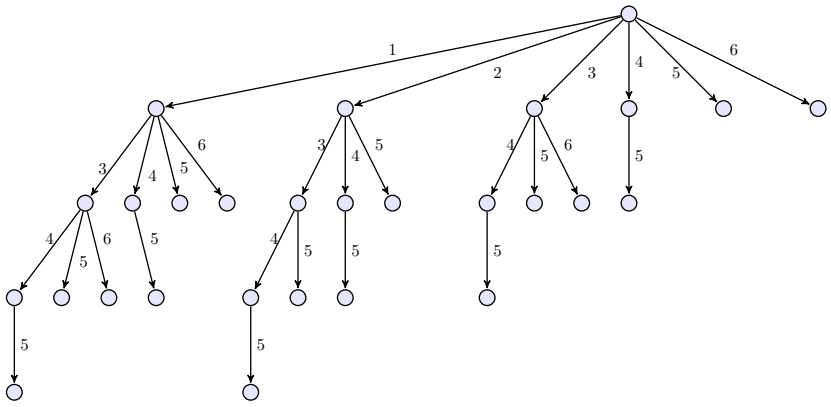

Fig. 5: Simplex Automaton of the simplicial complex in Figure 1.

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 [24] 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 $\mathcal{M}$. For any $K \in \mathcal{K}_\theta(n, k, d, m)$, let us define the Minimal Simplex Automaton ($\mathcal{M}$(SA)) as the minimal deterministic automaton which recognizes the language L_{K_θ}. Compressing ST can be seen as DFA minimization since merging identical subtrees corresponds to merging indistinguishable states in the automaton. It is possible to get $\mathcal{C}$(ST) from $\mathcal{M}$(SA) by duplicating the states such that for each node, the labels of all its incoming edges are the same, and then by moving the labels from the edges to the next node. Also, it should be observed that the number of edges in $\mathcal{C}$(ST) and the number of transitions in $\mathcal{M}$(SA) may not be the same. The reason is that states in $\mathcal{M}$(SA) having identical set of outgoing paths can merge even when the incoming set of transitions are different for each of these states, while such nodes in $\mathcal{C}$(ST) would not have merged.

Algorithmic aspects of DFA minimization have been well studied. For instance, Hopcroft's algorithm [19] 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 running time is shown in [19] 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 [25]. Also, in Appendix A we describe an adapted Hopcroft's algorithm to optimally compress ST. In Figure 6, we give the minimal automaton for the simplicial complex of Figure 1.

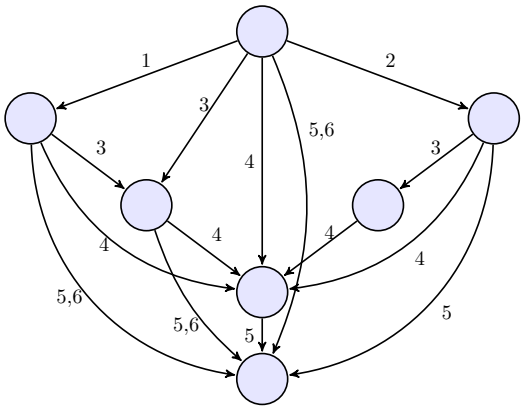

Fig. 6: Minimal Simplex Automaton of the simplicial complex in Figure 1.

While there are delicate differences between $\mathcal{M}(\text{SA})$ and $\mathcal{C}(\text{ST})$, we will see below that performing basic operations on $\mathcal{M}(\text{SA})$ is not very different from the way it is done for $\mathcal{C}(\text{ST})$.

3.2.1 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 all static operations i.e., all operations where we don't change the $\mathcal{M}(\text{SA})$ 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 earlier for $\mathcal{C}(\text{ST})$ in subsection 3.1.

Addition and deletion of simplexes 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 cofaces, edge contraction and elementary collapses.

3.2.2 Complexity Measure of Size for the Minimal Simplex Automaton

Our complexity measure in the paper would be minimizing SA to obtain $\mathcal{M}(\text{SA})$ with minimum number of states. We know from Myhill-Nerode theorem that there is a unique minimal DFA. Thus, if there was an automaton with less transitions than $\mathcal{M}(\text{SA})$ then, it should have more states than $\mathcal{M}(\text{SA})$. Let A be an automaton with a minimal number of transitions and let us run Hopcroft's algorithm on A. Since, at no point in Hopcroft's algorithm, we increase the number of transitions, the output has to be an automaton whose numbers of states and transitions are both minimal. It follows that the unique automaton output by Hopcroft's algorithm must have both a minimal number of states and a minimal number of transitions. This is proved more formally as Proposition 1 in [22].

In the rest of the paper, **we denote by** $|\text{SA}|$ **and** $|\mathcal{M}(\text{SA})|$**, the number of states in** SA **and** $\mathcal{M}(\text{SA})$ **respectively**. While we consider the number of states as a complexity measure of the size of SA and $\mathcal{M}(\text{SA})$, we will still use the number of edges as a complexity measure of the size of ST and $\mathcal{C}(\text{ST})$ because the bounds obtained relating the number of edges in $\mathcal{C}(\text{ST})$ and the number of nodes in $\mathcal{C}(\text{ST})$ are not satisfactory. The size of $\mathcal{M}(\text{SA})$ will be

discussed in detail 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* MxST(K) as an induced subgraph of ST(K). All leaves in the Simplex Tree corresponding to maximal simplices and the nodes encountered on the path from the root to these leaves are kept in the Maximal Simplex Tree and the remaining nodes are removed. MxST(K) is constructed as follows. We start from an empty tree and then insert the words representing the maximal simplices of K. Specifically, 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 Maximal Simplex Tree. We then append to the node representing $[\ell_0, \cdots, \ell_i]$ a path consisting of the nodes storing labels $\ell_{i+1}, \cdots, \ell_j$. Figure 7 shows the MxST of the simplicial complex given in Figure 1. In 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 MxST(K).

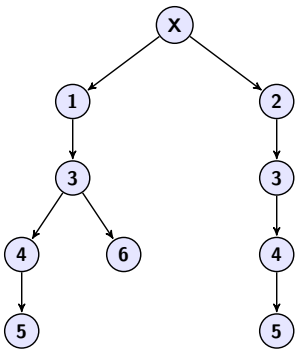

Fig. 7: Simplicial Complex of Figure 1 represented using Maximal Simplex Tree.

By the above construction of MxST, we add at most $d + 1$ nodes per maximal simplex. Hence, 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** |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.

4.1 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 (details of this analysis is provided in Appendix B) and note that it is already better than ST for some operations.

Operation	Cost
Identifying maximal cofaces of a 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	$\mathcal{O}(kdd_\sigma \log n)$
Edge Contraction	$\mathcal{O}(kd(k + \log n))$

Table 1: Cost of performing basic operations on MxST.

Moreover, we can augment the structure of MxST without paying for a lot of extra memory 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.

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

Proof The proof is by induction on the number n of vertices. When $n = 1$, we have two simplices (including the empty simplex) and one leaf. Now assume (induction hypothesis) that for all simplicial complexes on i vertices the corresponding ST has at least half of its nodes as leaves. Consider a ST on the vertex set $\{1, 2, ..., i + 1\}$ containing m simplices. Consider the subtree

under node 1 say ST_1. ST_1 represents a simplicial complex on the vertex set $\{2, 3, ..., i+1\}$ with node 1 acting as the root. Suppose ST_1 has m_1 nodes. Then, by the induction hypothesis, ST_1 has at least $\frac{m_1}{2}$ leaves. Now consider the rest of ST which can also be independently seen as a Simplex Tree ST_2 on the vertex set $\{2, ..., i+1\}$. Again, by the induction hypothesis, ST_2 has at least $\frac{m-m_1}{2}$ leaves. Thus ST has at least $\frac{m}{2}$ leaves. □

Differently from ST, $\mathcal{M}(\mathrm{SA})$ has only one leaf. The following lemma shows that $\mathcal{M}(\mathrm{SA})$ has at most half the number of nodes of ST plus one (follows directly from Lemma 1).

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

Proof Since there are m nodes in ST, SA has exactly m states and at least $\frac{m}{2}$ of them are leaves (a state without outgoing transitions). The $\frac{m}{2}$ leaves can be merged. Consequently, the number of states of $\mathcal{M}(SA(K))$ is at most $m - \frac{m}{2} + 1 = \frac{m}{2} + 1$. □

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

Lemma 3 *For any **pure** simplicial complex $K \in \mathcal{K}(n, k, d, m)$, $|\mathcal{M}(\mathrm{SA}(K))| \geq |\mathcal{M}(\mathrm{MxSA}(K))|$.*

Proof Let K be a pure simplicial complex. Let s_i be the initial state and S_f be the set of the states of outdegree zero. To each state we associate its depth, which is the the length of the longest directed path from s_i to this state.

We define another automaton B. The states of B are the states of $\mathrm{SA}(K)$. The state s_i is still the initial state and, the new final states are $\{s \in S_f \mid \mathrm{depth}(s) = d+1\}$. Finally, there is a transition between states u and v labeled by a if and only if this transition exists in $\mathrm{SA}(K)$ and if $\mathrm{depth}(v) = \mathrm{depth}(u) + 1$. Let us prove that B recognizes exactly the maximal simplices of K.

Let w be a word recognized by B. Then w is a word of length $d+1$ which was recognized by $\mathrm{SA}(K)$. Then, it corresponds to a simplex of K of dimension d. It is a maximal simplex of K.

On the other direction, let σ be a maximal simplex of K. Hence the word σ is accepted by $\mathrm{SA}(K)$ and of length $d+1$. If any transition which appears during this detection appears also in B then, σ is also accepted by B. Thus, let us assume for a contradiction that during the detection of the word σ, the simplex automaton $\mathrm{SA}(K)$ uses a transition from a state u to a state v such that $\mathrm{depth}(v) \neq \mathrm{depth}(u) + 1$ (let us choose the first transition where it happens). By definition of the depth of a state, we get $\delta_1 = \mathrm{depth}(v) > \mathrm{depth}(u) + 1 = \delta_2 + 1$. This means that there exists a word w of length δ such that by reading w from the initial state, we arrive into v. Then, w, σ_s (where σ_s is the suffix of σ of length $d - \delta_2$) is also accepted by $\mathrm{SA}(K)$. Consequently K contains a simplex of dimension $d + \delta_1 - \delta_2 - 1 > d$ and we have reached a contradiction. □

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 3 holds only for pure simplicial complexes because, if all complexes were allowed then, we will have complexes like in Example 1 where $|\mathcal{M}(\mathrm{SA}(K))| < |\mathcal{M}(\mathrm{MxSA}(K))|$.

Example 1 Consider the simplicial complex on seven 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 would like to analyze two possible sources of compression in ST. A first type of compression may happen when a simplex σ belongs to several maximal simplices and its vertices appear as the last vertices of those maximal simplices. Then compression will factorize σ so that it will appear only once as a common suffix of several words. A second type of compression occurs when considering a single maximal simplex. Here too, ST stores many different words with common suffixes and compression will factorize these common suffixes. Intuitively, the first type captures compression solely in MxST (i.e., because of the input and the labeling on vertices we have defined) and the second source analyzes possible compression because of the rich structure of ST. Now, we will see a result which guarantees compression for pure simplicial complexes regardless of the labeling of the vertices:

Lemma 4 *For any* ***pure*** *simplicial complex $K \in \mathcal{K}(n,k,d,m)$, we have that $|\mathcal{M}(\mathrm{SA})|$ is always less than $|\mathrm{SA}|$ when $k < d$ and $d \geq 2$.*

Proof Let K be a pure simplicial complex. Let $\sigma = v_1, \ldots, v_d, v_{d+1}$ be a maximal simplex. Let us define $\nu = \{v_1, \ldots, v_{d-1}\}$ and, $M = \{m \in K \mid m \text{ is maximal and } v_d \in m\}$. We also define $P_\nu \subseteq \mathcal{P}(\nu)$ as the projection of M on to ν, which is more formally written as:

$$P_\nu = \{a \subseteq \nu \mid \exists m \in M, a = m \cap \nu\}.$$

We notice that $\nu \in P_\nu$. Since $d > k \geq |M| \geq |P_\nu|$, it follows that there exists $b \subseteq \nu$ such that $|b| = |\nu| - 1$ and which is not in P_ν. Let s_ν and s_b be the states in $\mathrm{SA}(K)$ reached by reading the words $\nu \cup v_d$ and $b \cup v_d$. As the language is closed by subwords and as $b \subseteq \nu$, any accepting word from the state s_ν is also an accepting word from the state s_b. Reciprocally, if w is an accepting word from the state s_b then, $b \cup v_d \cup w$ is a face of a maximal simplex m. The projection of m on ν contains b and by definition of b is strictly larger. Hence $\nu \subseteq m$, and so, $\nu \cup v_d \cup w \in K$. Consequently the states s_ν and s_b are equivalent and can be merged. □

In fact, the above result is close to tight: in Example 4, we have a pure simplicial complex with $k = d$ and $|\mathrm{ST}| = |\mathcal{C}(\mathrm{ST})|$. We remark here that in cases of impossibility of compression we will analyze the compression of ST rather than the minimization of SA 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 2 says otherwise.

Example 2 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 $|\mathcal{C}(\mathrm{MxST})| < |\mathrm{MxST}|$ does not imply $|\mathcal{C}(\mathrm{ST})| < |\mathrm{ST}|$ as can be seen in Example 3.

Example 3 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 both Examples 2 and 4, we saw simplicial complexes 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 5 *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 $|\mathcal{M}(\mathrm{SA}(K_{\pi\circ\theta}))| < |\mathrm{SA}(K_{\pi\circ\theta})|$.*

Proof Let $m = v_{\ell_0} \cdots v_{\ell_d}$ be a maximal simplex in K_θ. We construct π by swapping ℓ_{d-1} and ℓ_d with $n - 1$ and n respectively. In $\mathrm{ST}(K_{\pi\circ\theta})$, note that the node under root with label $n - 1$ and the node corresponding to simplex $v_{\pi(\ell_0)} \cdots v_{\pi(\ell_{d-1})}$ are identical and thus can be merged. □

We would have liked to obtain better bounds for the size of $\mathcal{M}(\mathrm{SA})$ 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 one is given by the ratio of |ST| and $|\mathcal{C}(\text{ST})|$ and the second by the ratio of |MxST| and $|\mathcal{C}(\text{MxST})|$. All experiments performed below record the extent of compression of ST. Ideally, we would have liked to record the extent of minimization of SA since $\mathcal{M}(\text{SA})$ is more compact than $\mathcal{C}(\text{ST})$. Unfortunately, this has not been possible due to the lack of available libraries able to handle very large automata. The results below for $\mathcal{C}(\text{ST})$ are nonetheless positive, substantiating our claim that compression of ST leads to a compact data structure.

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

No	n	r	d	k	$\lvert\text{ST}\rvert = m-1$	$\lvert\text{MxST}\rvert$	$\lvert\mathcal{C}(\text{ST})\rvert$	ρ_{ST}	$\lvert\mathcal{C}(\text{MxST})\rvert$	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 2: Analysis of experiments on Data Set 1.

First, observe that |MxST| is considerably smaller than |ST|. This is expected, as it is likely that k is polynomially related to n for Rips complexes. Also, while we observe insignificant compression in MxST, ρ_{ST} increases rapidly as r is increased. This indicates that compression strongly exploits the combinatorial redundancy of ST (i.e., storing each simplex explicitly through a node) and works particularly well for the Simplex Tree.

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 this 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, $|\mathcal{C}(\text{ST})|$ and $|\mathcal{C}(\text{MxST})|$ for the various complexes constructed.

No	n	p	d	k	$\lvert\text{ST}\rvert = m-1$	$\lvert\text{MxST}\rvert$	$\lvert\mathcal{C}(\text{ST})\rvert$	ρ_{ST}	$\lvert\mathcal{C}(\text{MxST})\rvert$	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

Table 3: Analysis of experiments on Data Set 2.

Here we observe staggering values for ρ_{ST} which increases as the simplicial complex grows larger. This is primarily because random simplicial complexes don't behave like pathological simplicial complexes (such as Examples 2 and 4) which hinder compression.

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* 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 can be obtained by modifying MxST or can be constructed from the maximal simplices of 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.

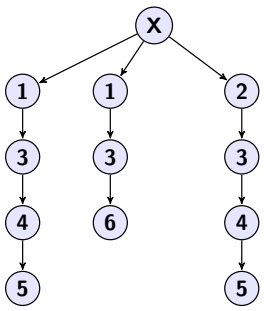

Fig. 8: Unprefixing the Maximal Simplex Tree of Figure 7.

2. **Transitive Closure** ($\mathcal{T}$)**:** For every pair of nodes (u, v) in $\mathcal{U}$(MxST) (u not being the root), if there is a path from u to v then, add an edge from u to v in $\mathcal{T}(\mathcal{U}(\text{MxST}))$ (if it doesn't already exist).
3. **Expanding Representation**($\mathcal{R}$)**:** For every node v in $\mathcal{T}(\mathcal{U}(\text{MxST}))$ 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 starting from the children of the root and recursively moving down to children of smallest label. As a demonstration, in Figure 10, we show the result of expanding *one* node in the $\mathcal{T}(\mathcal{U}(\text{MxST})$ of Figure 9.

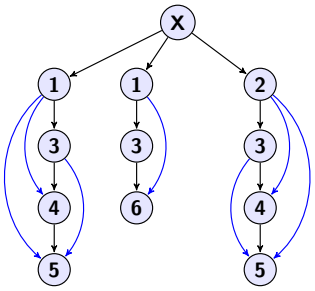

Fig. 9: Transitive Close of the Unprefixed Maximal Simplex Tree of Figure 8.

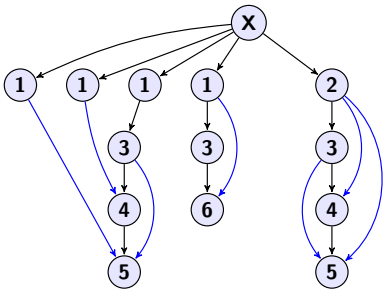

Fig. 10: Expanding **one** node in $\mathcal{T}(\mathcal{U}(\text{MxST})$ of Figure 9.

$\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST})))$ is the Simplex Array List. From the construction of SAL, it is clear that each node in SAL uniquely represents an edge in the simplicial complex. Figure 11 shows SAL representation of the simplicial complex given in Figure 1.

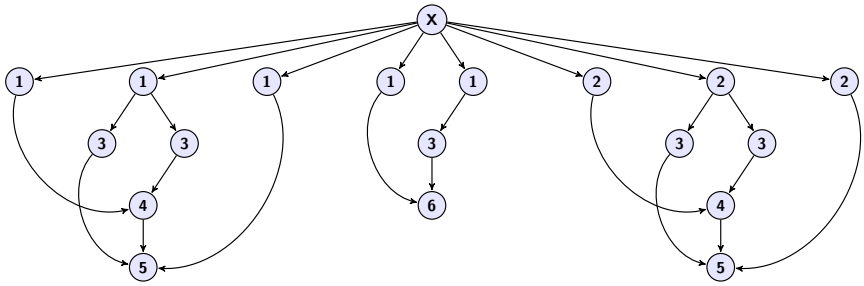

Fig. 11: Simplicial Complex of Figure 1 represented using $\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST})))$.

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 (cf. Figure 12 for an example):
$S_1 = \{(\ell_i, \ell_{i'}, \mathcal{H}(\sigma)) \mid i \in \{0, 1, \ldots, j-1\}, i' \in \{i+1, \ldots, j\}\}$
$S_2 = \{(\ell_j, \varphi, \mathcal{H}(\sigma))\}$
where φ denotes an empty label. 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 $i \in \{1, 2, \ldots, n\}$, 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.

The resultant graph obtained after removing the root in $\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST})))$ 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.

We remark here that we use hash function $\mathcal{H}$ to generate keys for simplices because it is an efficient way to reuse keys (in case of multiple insertions and removals).

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 $k\left(\frac{d^2(d+1)}{2}+d\right)$. Further, in each node we store the labels of two vertices (which requires $\log n$ bits) and a hashed value (which requires $\log k$ bits). Hence, the space required to store SAL(K) is $\mathcal{O}(kd^2(d + \log n + \log k))$. **Also, unless otherwise stated** |SAL| **refers to number of edges in** SAL**.** Since SAL is constructed from $\mathcal{U}$(MxST), we have the following lemma:

Lemma 6 *The number of nodes and edges in SAL are both invariant over the labeling of the vertices in the simplicial complex.*

Intuitively, SAL is representing K by storing all the edges of K explicitly as nodes in SAL(K) and the edges in SAL(K) are used to capture the incidence relations between simplices. More precisely, a path of length j in SAL(K) corresponds to a unique j-simplex in K. 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

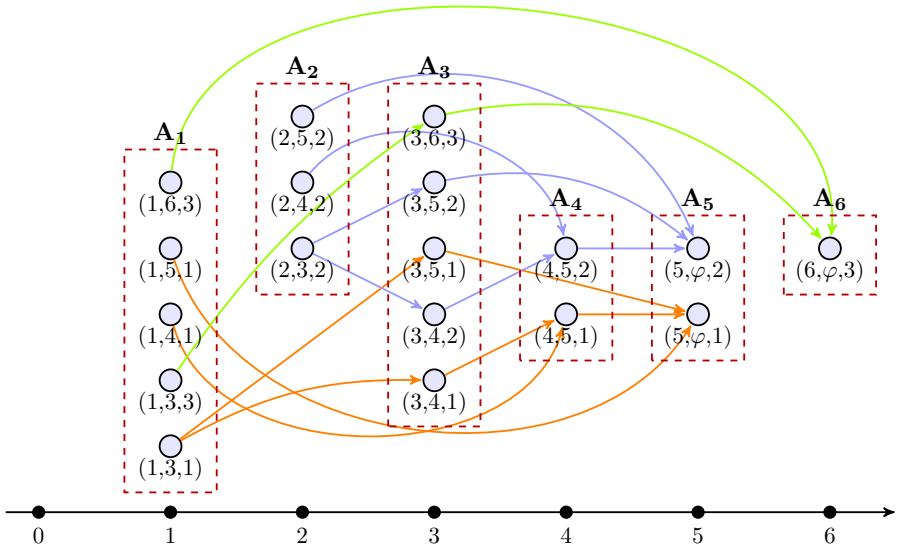

Fig. 12: Simplex Array List for complex in Figure 1 embedded on the number line.

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 7 *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.*

Proof For any path in SAL(K), it belongs to a connected component of SAL(K). In particular, there exists a maximal simplex m such that all nodes in this component are of the form $(a, b, \mathcal{H}(m))$ for some a, b such that v_a is a vertex of m. Consequently, all vertices read during this path belong to m, it means that the corresponding simplex is a face of m, so it belongs to K.

On the other hand, if $\sigma = v_{\ell_1} \cdots v_{\ell_r}$ is a simplex of K, it belongs to some maximal simplex $m = v_{\ell'_1} \cdots v_{\ell'_{d_m}}$ where we have $\{\ell_1, \ell_2, \ldots, \ell_r\} \subseteq \{\ell'_1, \ldots, \ell'_{d_m}\}$. In SAL($K$), we look at the connected component associated with m. While we can associate any node with label (a, b, z) uniquely to edge $a - b$, we can also uniquely identify to vertex a. We associate σ to the path:

$$(\ell_1, \ell_2, \mathcal{H}(m)) - (\ell_2, \ell_3, \mathcal{H}(m)) - \cdots - (\ell_r, x, \mathcal{H}(m)),$$

for some $x \in \{\ell_r + 1, \ldots, n, \varphi\}$. The existence of the path is confirmed because of the way we introduced edges in the connected component. □

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 4.

Example 4 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 $\mathrm{SAL}(K)$ has $\frac{k^2(k+1)}{2} + k$ nodes while $\mathcal{M}(\mathrm{SA}(K))$ 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 $\mathcal{M}(\mathrm{SA})$, as the gap in their sizes can be exponential.

Building $\mathrm{SAL}(K)$ can be seen as partially compressing the Simplex Tree $\mathrm{ST}(\sigma)$ associated to each maximal simplex σ (where σ and its subfaces are seen as a subcomplex). Compressing $\mathrm{ST}(\sigma)$ will lead to a subtree which is exactly the same as the transitive closure of $\mathrm{MxST}(\sigma)$. Therefore, collecting all $\mathcal{C}(\mathrm{ST})(\sigma)$ for all maximal simplices σ and merging the roots is the same as $\mathcal{T}(\mathcal{U}(\mathrm{MxST}(K)))$. Now applying $\mathcal{R}$ on $\mathcal{T}(\mathcal{U}(\mathrm{MxST}(K)))$ 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 subarrays 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 7 there should exist a path corresponding to this simplex which would imply $\bigcap_i P(B_{\ell_i}) \neq \emptyset$, and if $\tau \in \bigcap_i P(B_{\ell_i})$ then, σ is a face of τ. 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, the total running time is $\mathcal{O}(d_\sigma(\gamma \log \zeta + \log \zeta)) = \mathcal{O}(d_\sigma \Gamma_1 \log(\Gamma_0 d)) = \mathcal{O}(d_\sigma \Gamma_1 \log(kd))$.

For example, consider the SAL of figure 12 and we have to check the 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 $\bigcap_i P(B_i) = \{2\} \neq \emptyset$, 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 exists 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 $\mathcal{O}(d_\sigma^3 \Gamma_1 \log(\Gamma_0 d))$). 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 time at most $\mathcal{O}(d_\sigma^4 \Gamma_1)$. Therefore, total time for insertion is $\mathcal{O}(d_\sigma^3 \Gamma_1 (d_\sigma + \log(\Gamma_0 d))) = \mathcal{O}(d_\sigma^3 \Gamma_1 (d_\sigma + \log(kd)))$.

Removal. To remove a face σ, obtain the maximal simplices which contain it (can be done through a membership query in $\mathcal{O}(d_\sigma \Gamma_1 \log(\Gamma_0 d))$ time). Next, remove the above maximal simplices and then insert the facets of the above maximal simplices which do not contain σ. More precisely, for each of the above obtained maximal simplices make d_σ copies of the corresponding 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 $\mathcal{O}(d_\sigma d^3 \Gamma_1 \log(\Gamma_0 d)) = \mathcal{O}(d_\sigma d^3 \Gamma_1 \log(kd))$.

Elementary Collapse. A simplex τ is collapsible through one of its faces σ, if τ is the only coface of σ. Such a pair (σ, τ) is called a free pair, and removing both faces of a free pair is an elementary collapse. Given a pair of simplices (σ, τ), to check if it is a free pair is done by obtaining the list of all maximal simplices which contain σ, through the membership query (costs $\mathcal{O}(d_\sigma \Gamma_1 \log(\Gamma_0 d))$ time) and then checking if τ is the only member in that list. If yes, remove τ and insert the facets (except σ) which are now maximal after removing τ. This takes time $\mathcal{O}(d_\sigma^4 + d_\sigma^2 \Gamma_1 \log(\Gamma_0 d))$. Thus, the total running time is $\mathcal{O}(d_\sigma^2 (d_\sigma^2 + \Gamma_1 \log(\Gamma_0 d))) = \mathcal{O}(d_\sigma^2 (d_\sigma^2 + \Gamma_1 \log(kd)))$.

Edge Contraction. Here we cannot do better than entirely rebuilding the parts of SAL corresponding to maximal simplices which contain the vertices in the edge to be contracted and therefore the cost of the operation is $\mathcal{O}(\Gamma_0 (d(\Gamma_1 \log(kd) + d^2)))$, as the number of maximal simplices a vertex may be

part of is at most Γ_0 (by definition) and checking if a simplex remains maximal after the edge contraction can be done through the membership query.

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

	ST	MxST	SAL
Storage	$\mathcal{O}(k2^d \log n)$	$\mathcal{O}(kd \log n)$	$\mathcal{O}(kd^2(d + \log n + \log k))$
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 simplex σ	$\mathcal{O}(2^{d_\sigma} d_\sigma \log n)$	$\mathcal{O}(kd \log n)$	$\mathcal{O}(d_\sigma^3 \Gamma_1 (d_\sigma + \log(kd)))$
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	$\mathcal{O}(2^{d_\sigma} \log n)$	$\mathcal{O}(kdd_\sigma \log n)$	$\mathcal{O}(d_\sigma^2 (d_\sigma^2 + \Gamma_1 \log(kd)))$
Edge Contraction	$\mathcal{O}(md)$	$\mathcal{O}(kd(k+\log n))$	$\mathcal{O}(\Gamma_0 (d(\Gamma_1 \log(kd) + d^2)))$

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

Performance of SAL. Plainly, if the number of maximal simplices is small (i.e., can be considered 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 $\mathcal{O}(kd^2(d + \log n + \log k))$ 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.

No	n	r	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

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

It is interesting to note that the size of SAL is larger than the size of $\mathcal{C}(\text{ST})$ but much smaller than the size of ST. This is expected, as SAL promises to perform most 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, which is supported by the above results. Also, we note that for larger simplicial complexes such as complexes No 3 and 4, there is a noticeable 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. It is worth noting 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 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 sensitive to the local structure of the complex.

6.4 A Sequence of Representations for Simplicial Complexes

We can use the operation $\mathcal{R}$ to generate a sequence of data structures, each more powerful than the previous ones (but also bulkier). More formally, consider the sequence of data structures $\langle \Lambda \rangle$, where $\Lambda_{-1} = \text{MxST}$, $\Lambda_0 = \mathcal{U}(\text{MxST})$ and $\Lambda_i = \mathcal{R}^i(\mathcal{T}(\mathcal{U}(\text{MxST})))$, for all $i \in \mathbb{N}$. Note that $\Lambda_1 = \text{SAL}$. Further, for all $i \in \mathbb{N}$, we will refer to the data structure Λ_i by the name i-SAL (we will continue to refer to 1-SAL as SAL).

Further, we see that in the i^{th} element of the sequence, every node which is not a leaf (sink) in the data structure corresponds to a unique i-simplex in the simplicial complex. Also for all i-SAL, $i \in \mathbb{N}$, we have that it is a NFA recognizing all the simplices in the complex. As we move along the sequence, the size of the data structure blows up by a factor of d at each step. But in return, we gain efficiency in searching for simplices as the membership query depends on Γ_i which decreases as i increases.

In section 6.4.1, we will see how to construct 0-SAL (i.e., $\mathcal{U}(\text{MxST})$) from the maximal simplices of a simplicial complex and in section 6.4.2, we will see how to construct $\mathcal{R}(\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST})))) = 2\text{-SAL}$ from the maximal simplices of a simplicial complex, and this would help to demonstrate the construction of data structures which appear later in the sequence $\langle \Lambda \rangle$.

6.4.1 Unprefixed Maximal Simplex Tree

The *Unprefixed Maximal Simplex Tree* $\mathcal{U}$(MxST) or 0-SAL is a directed acyclic graph which can be obtained by modifying MxST or can be constructed from the maximal simplices of K. We will describe the latter here. We initially have n empty arrays $A_1, \ldots, A_n$ and for every 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 insert $\mathcal{H}(\sigma)$ in the arrays $A_{\ell_0}, \ldots, A_{\ell_j}$. Thus determining membership of a simplex again reduces to computing set intersection and insertion and removal of simplices are performed as in MxST but here we are equipped with a faster search operation (i.e., more efficient membership query). We summarize in Table 6 the asymptotic cost of basic operations for 0-SAL and compare it with MxST.

Operation	Cost for MxST	Cost for 0-SAL
Membership of a simplex σ	$\mathcal{O}(kd \log n)$	$\mathcal{O}(d_\sigma \Gamma_0 \log k)$
Insertion of a simplex σ	$\mathcal{O}(kd \log n)$	$\mathcal{O}(d_\sigma \Gamma_0 (d_\sigma + \log k))$
Removal of a face	$\mathcal{O}(kd \log n)$	$\mathcal{O}(d_\sigma d \Gamma_0 \log k)$
Elementary Collapse	$\mathcal{O}(kdd_\sigma \log n)$	$\mathcal{O}(d_\sigma^2 \Gamma_0 \log k)$
Edge Contraction	$\mathcal{O}(kd(k + \log n))$	$\mathcal{O}(\Gamma_0^2 d \log k)$

Table 6: Cost of performing basic operations on 0-SAL which is compared with cost of such operations on MxST.

1-SAL has two advantages over 0-SAL. First, 1-SAL stores all simplices through its paths (Lemma 7) and this may help in storing filtrations. In addition, it is likely that Γ_1 is significantly smaller than Γ_0.

6.4.2 2-Simplex Array List

The *2-Simplex Array List* 2-SAL(K) is a directed acyclic graph which can be obtained by modifying MxST (i.e., 2-SAL = $\mathcal{R}(\mathcal{R}(\mathcal{T}(\mathcal{U}(\text{MxST}))))$) or can be constructed from the maximal simplices of K. We will describe the latter here. 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 $\frac{j(j^2+5)}{6} + 1$ new nodes in 2-SAL. We build a set of $\frac{j(j^2+5)}{6} + 1$ labels and assign uniquely a label to each node. The set of labels is defined as the union of the following three sets (cf. Figure 13 for an example):
$S_1 = \{(\ell_{i_1}, (\ell_{i_2}, \ell_{i_3}), \mathcal{H}(\sigma)) \mid i_1 \in \{0, 1, \ldots, j-2\}, i_2 \in \{i_1 + 1, \ldots, j-1\}, i_3 \in \{i_2 + 1, \ldots, j\}\}$
$S_2 = \{(\ell_i, (\ell_j, \varphi), \mathcal{H}(\sigma)) \mid i \in \{0, 1, \ldots, j-1\}\}$
$S_3 = \{(\ell_j, (\varphi, \varphi), \mathcal{H}(\sigma))\}$

where φ denotes an empty label. Now introduce an edge between two nodes with labels $(\ell_{i_1}, (\ell_{i_2}, \ell_{i_3}), \mathcal{H}(\sigma))$ and $(\ell_{i_4}, (\ell_{i_5}, \ell_{i_6}), \mathcal{H}(\sigma))$ if and only if $i_2 = i_4$ and $i_3 = i_5$. Next, introduce an edge between two nodes with labels $(\ell_{i_1}, (\ell_{i_2}, \ell_{i_3}), \mathcal{H}(\sigma))$ and $(\ell_{i_4}, (\ell_j, \varphi), \mathcal{H}(\sigma))$ if and only if $i_2 = i_4$ and $i_3 = j$. Finally, introduce an edge from every node with label in S_2 to the node with label in S_3. Thus, in 2-SAL we represent a maximal j-simplex using a connected component containing $|S_1| + |S_2| + |S_3| = \frac{j(j^2+5)}{6} + 1$ nodes. To perform basic operations efficiently, embed 2-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 $(i, (a, b), z)$ for some $z \in \{1, \ldots, k\}$ and $a, b \in \{i+1, \ldots, n, \varphi\}$. Sort each A_i based on a and in case of ties, sort them based on b and in case of further ties sort them based on z.

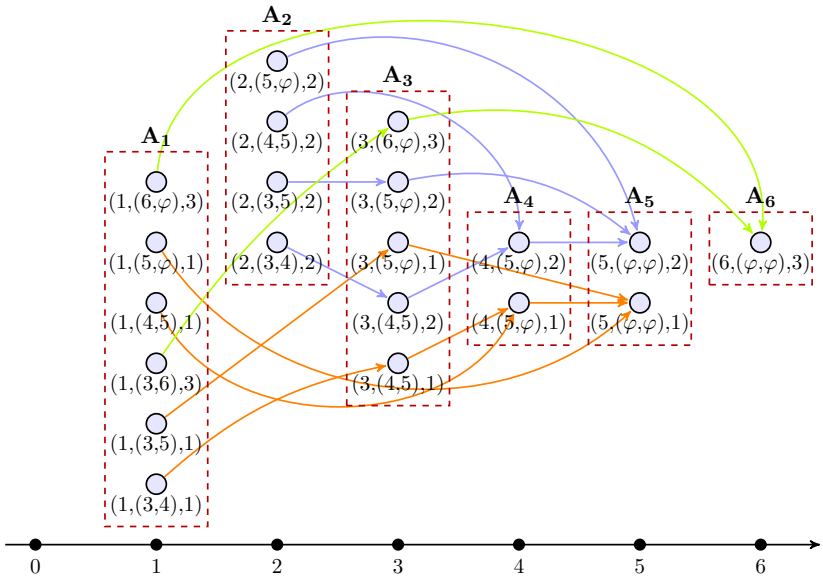

Fig. 13: 2 - Simplex Array List for complex in Figure 1 embedded on the number line.

2-SAL(K) has at most $k\left(\frac{d(d^2+5)}{6} + 1\right)$ nodes and, for each maximal simplex of dimension d_σ, the outdegree of any node in the connected component is at most d_σ. Therefore, the total number of edges in 2-SAL(K) is at most $k\left(\frac{d^2(d^2+5)}{6} + d\right)$. Hence, the space required to store 2-SAL(K) is $\mathcal{O}(kd^3(d + \log n + \log k))$.

We summarize in Table 7 the asymptotic cost of basic operations for 2 - SAL and compare it with SAL. All the basic operations are performed similar

to the way we did in SAL except that here we have more structure and thus searching becomes more efficient (Γ_1 is replaced by the smaller Γ_2) but pay extra in size because we have to maintain the additional structure.

Operation	Cost for SAL	Cost for 2-SAL
Membership of a simplex σ	$\mathcal{O}(d_\sigma(\Gamma_1 \log(kd)))$	$\mathcal{O}(d_\sigma \Gamma_2 \log(kd^2))$
Insertion of a maximal simplex σ	$\mathcal{O}(d_\sigma^3(\Gamma_1 d_\sigma + \log(kd)))$	$\mathcal{O}(d_\sigma^3 \Gamma_2(d_\sigma^3 + \log(kd^2)))$
Removal of a face	$\mathcal{O}(d_\sigma d^3(\Gamma_1 + \log(kd)))$	$\mathcal{O}(d_\sigma d^4 \Gamma_2 \log(kd^2))$
Elementary Collapse	$\mathcal{O}(d_\sigma^2(d_\sigma^2 + \Gamma_1 + \log(kd^2)))$	$\mathcal{O}(d_\sigma^2(d_\sigma^3 + \Gamma_2 \log(kd^2)))$
Edge Contraction	$\mathcal{O}(\Gamma_0(d(\Gamma_1 \log(kd) + d^2)))$	$\mathcal{O}(\Gamma_0(d(\Gamma_2 \log(kd^2) + d^3)))$

Table 7: Cost of performing basic operations on 2-SAL which is compared with cost of such operations on SAL.

6.4.3 The Bottom Line

A natural question to resolve is which element in $\langle \Lambda \rangle$ should one pick for representing a simplicial complex. This indeed depends on the nature of data and the type of complex. For instance, consider the case of a Rips complex whose vertex set is a good sample of a smooth manifold of low intrinsic dimension. Then, we would settle for either Λ_0 or Λ_1 as we expect Γ_0 itself to be quite low. However if the simplicial complex is of high dimension and if there are central simplices on which most maximal simplices are built on then, it might be better to look at elements higher up in the sequence as Γ might be quite high in the beginning and can collapse quickly after some point.

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, the size of both ST and SAL are invariant over the labeling of vertices in a simplicial complex (see Lemma 6). However, this is not the case with MxST and $\mathcal{M}$(SA). 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.

In all of the hardness results we will see, the reduction will be from vertex cover for some specific graphs, where graphs are considered as 1-dimensional simplicial complexes. More formally, given a graph G on vertex set V and edge set E, the associated 1-dimensional simplicial complex is defined on the vertex set V which admits all edges in E as 1-dimensional simplices. When there is no confusion, we will refer to this simplicial complex also as G.

We observe that MxST(G) is of height 2. In layer 0 we have the root, and in layer 2 we have all the leaves (we may have some leaves in layer 1 if there are vertices in G of degree zero). Since the number of leaves in the MxST is equal to the number of maximal simplices in G which is in turn equal to the number of edges in G (plus the number of zero-degree vertices), regardless of the ordering on vertices in G, the number of nodes in layer 0 and 2 are fixed to 1 and $|E|$ respectively. Further note that vertices on layer 1 form a vertex cover of G.

We will similarly analyze $\mathcal{C}(\text{ST}(G))$. ST($G$) can be obtained from MxST(G) by introducing nodes on layer 1 (and edges from the root to these nodes) such that all vertices appear in nodes of layer 1 of ST(G). This means that in ST(G), we have the root in layer 0, $|V|$ nodes on layer 1, and $|E|$ nodes on layer 2. We recapitulate here that the size of ST is invariant over the ordering of vertices (as can be observed above – $|\text{ST}(G)| = |V| + |E|$). More importantly, for 1 - dimensional simplicial complexes we have that $|\mathcal{C}(\text{ST})| = |\text{ST}|$. Therefore to analyze the impact of labeling on the size of $\mathcal{C}$(ST), we will have to move to higher dimensional complexes (which we indeed do).

7.1 Optimal Labeling for the Maximal Simplex Tree

The size of MxST is very sensitive to the labeling of the vertices. For instance, in Example 5, by reversing the labeling of vertices, we increase the size of MxST(K) by a factor of order k.

Example 5 Let $K \in \mathcal{K}(d+k, k, d, m)$ whose set of maximal simplices is $\{1, 2, \ldots, d, d+i | 1 \leq i \leq k\}$.

First, we formalize the label ordering problem on MxST: 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 $|\text{MxST}(K_{\pi \circ \theta})| \leq \alpha$? Let us refer to this problem as MxSTMINIMIZATION(K_θ, α) and from Theorem 17 of [9], we know that it is NP-Complete even for 1-dimensional complexes.

Theorem 2 ([9]) *MxSTMINIMIZATION is NP-Complete.*

Proof Clearly MxSTMINIMIZATION is in NP. We will now show that it is NP-Hard as well. Given a connected graph G on vertex set V (with θ being a labeling of the vertices from 1 to $|V|$) and edge set E, we define K_θ to be the 1-dimensional simplicial complex associated to G. Since G is connected, we have that all the leaves in MxST(K_θ) appear in layer 2. Thus finding a vertex cover for G of size at most α is equivalent to finding an ordering π on the vertices in K_θ (to determine which of these should appear in nodes in layer 1 of the MxST) such that $|\text{MxST}(K_{\pi \circ \theta})| \leq \alpha + |E|$. Since vertex cover problem is NP-Hard [20], we have that MxSTMINIMIZATION is also NP-Hard. Thus MxSTMINIMIZATION is NP-Complete. □

Intuitively, finding a good labeling for $\mathcal{C}(\text{MxST})$ seems harder than for MxST. More formally, 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 $|\mathcal{C}(\text{MxST}(K_{\pi\circ\theta}))| \leq \alpha$? Let us refer to this problem as CMxstMinimization(K_θ,α). Corollary 1 easily follows from Theorem 2 as, for any 1-dimensional simplicial complex and any fixed labeling, $|\mathcal{C}(\text{MxST})| = |\text{MxST}|$.

Corollary 1 *CMxstMinimization is NP-Complete.*

Proof It is clear that CMxstMinimization is in NP. We know from the proof of Theorem 2 that it is NP-Hard to decide MxstMinimization even for pure simplicial complexes of dimension 1. We will thus show a reduction from MxstMinimization for such simplicial complexes to CMxstMinimization. For any pure simplicial complex K of dimension 1 under any labelling θ of its vertices, the size of MxST and $\mathcal{C}(\text{MxST})$ for K_θ are the same. Thus any solution for an instance of MxstMinimization is also a solution for the same instance for CMxstMinimization and vice versa. □

7.2 Optimal Labeling for the Minimal Simplex Automaton

To prove that finding a good labeling for $\mathcal{M}(\text{SA})$ is hard, we use a reduction from another instance of vertex cover for a special class of graphs. We say a graph G is square-free if, for every two vertices u,v in G, the number of common neighbors in G is at most 1.

Lemma 8 *Vertex Cover problem on square-free graphs is NP-Hard.*

Proof We observe that in the reduction from SAT to 3-SAT (Theorem 3.1. of [17]), every clause has exactly three distinct variables (where x and $\bar{x}$ are considered as the same variable). Next, we observe that if every clause has exactly three distinct variables then, in the reduction from 3-SAT to Vertex Cover (Theorem 3.3. of [17]), the graph constructed does not have a cycle of length four. □

We will now formalize the decision problem for $\mathcal{M}(\text{SA})$. Given an integer α and $K_\theta \in \mathcal{K}_\theta(n,k,d,m)$, does there exist a permutation π of $1,2,\ldots,n$ such that $|\mathcal{M}(\text{SA})(K_{\pi\circ\theta})| \leq \alpha$? Let us refer to this problem as MsaMinimization(K_θ,α). We work with square-free graphs because by using such graphs for building simplicial complexes, we would overcome scenarios in which we have two states in $\mathcal{M}(\text{SA})$ with identical set of outgoing transitions but have different sets of incoming transitions.

Theorem 3 *MsaMinimization is NP-Complete.*

Proof It is clear that MsaMinimization is in NP. We will now show that it is NP-Hard as well. Given a square-free graph G on vertex set V and edge set

E, and a labeling θ from 1 to $|V|$ on the vertices, we define K_θ to be the 1-dimensional simplicial complexes associated to G. In the following paragraphs we will prove that G has a vertex cover of size at most α if and only if there exists an ordering π on the vertices of K_θ such that $|\mathcal{M}(\mathrm{SA}(K_{\pi\circ\theta}))| \leq \alpha + 2$.

First, we prove the forward direction. We define the height of a state as the length of the longest sequence of transitions from the initial state to that state. We define the height of an automaton as the maximum over the height of all states in the automaton. Thus $\mathcal{M}(\mathrm{SA}(K_\theta))$ is of height 2. At height 0 we have the initial state, and at height 2 we have a single state. Transitions from states in height 1 to height 2 correspond to the edges of G regardless of the ordering on the vertices in K_θ. Note that vertices at height 1 form a vertex cover of G. Thus if G has a vertex cover of size at most α then, we can construct an ordering π on the vertices of K_θ (by allowing all vertices appearing in the vertex cover to appear before the remaining vertices) such that $|\mathcal{M}(\mathrm{SA}(K_{\pi\circ\theta}))| \leq \alpha + 2$.

Now, we prove the reverse direction. Suppose there exists an ordering π on the vertices of K_θ such that $|\mathcal{M}(\mathrm{SA}(K_{\pi\circ\theta}))| \leq \alpha + 2$. If there is a state s in $\mathcal{M}(\mathrm{SA}(K_{\pi\circ\theta}))$ at height 1 which has more than one incoming transition then, it cannot have more than one outgoing transition because G is square-free. In this case, the outgoing transition of s is rewired to be the incoming transition from the initial state and the incoming transitions are rewired to be the outgoing transitions from s to the single state at height 2. Once this swapping (i.e., rewiring) is done for each state at height 1, we have ensured that the number of incoming transitions for such states is one (because the number of outgoing transitions from these states before rewiring was one). Additionally, we note that by performing the above rewiring in $\mathcal{M}(\mathrm{SA}(K_{\pi\circ\theta}))$ we have not introduced (or removed) any new states, and thus the size of $\mathcal{M}(\mathrm{SA}(K_{\pi\circ\theta}))$ has not changed (and it is computing the same language as before). Therefore, choosing the set of all labels of outgoing transitions from the initial state to states at height 1 gives a subset of the vertex set of size at most α and does indeed form a vertex cover of G. From Lemma 8 we know that vertex cover is NP-Hard for square-free graphs, which implies that MsaMinimization is also NP-Hard. Thus MsaMinimization is NP-Complete. □

We believe that to obtain reasonably small sized MxST and $\mathcal{M}(\mathrm{SA})$ one has to label vertices in decreasing order of $k_v \stackrel{\text{def}}{=}$ number of maximal simplices containing vertex v. Thus, in practice one may use this heuristic to find a good labeling.

7.3 Optimal Labeling for the Compressed Simplex Tree

We had observed in Section 3.2 the delicate relationship between the sizes of $\mathcal{M}(\mathrm{SA})$ and $\mathcal{C}(\mathrm{ST})$. Additionally, we have that the complexity measure of sizes for $\mathcal{M}(\mathrm{SA})$ and $\mathcal{C}(\mathrm{ST})$ are not coherent and thus, we will not be able to use Theorem 3 here to prove hardness result. On the other hand, we cannot

(trivially) extend the reduction from MxSTMINIMIZATION to that of finding good labelings for $\mathcal{C}(\mathrm{ST})$ because, as stated earlier, for 1-dimensional simplicial complexes we have $|\mathcal{C}(\mathrm{ST})| = |\mathrm{ST}|$. Instead we append the simplicial complex (associated to) G and construct 2-dimensional complexes which we then use to prove hardness result for finding good labelings for $\mathcal{C}(\mathrm{ST})$.

Let us first formalize the decision problem. 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 $|\mathcal{C}(\mathrm{ST}(K_{\pi\circ\theta}))| \leq \alpha$? Let us refer to this problem as CSTMINIMIZATION(K_θ,α). To prove the hardness result for CSTMINIMIZATION, we provide a reduction from a special instance of vertex cover problem which will be shown to be NP-hard. Specifically, we restrict vertex cover problem to a special class of graphs we call as the King-Maker graphs. A graph G is called a King-Maker graph if there exists two vertices u, v in G such that u is connected to all vertices in G (and thus we shall fondly refer to this vertex as the king) and v is of degree 1.

Lemma 9 *Vertex Cover problem on King-Maker graphs is NP-Hard.*

Proof Given a graph G on vertex set V and edge set E, we build a King-Maker graph G' by adding two new vertices u and v, and adding an edge between every vertex in G and u, and an edge between u and v. If G has a vertex cover of size α then, G' has a vertex cover of size $\alpha+1$ and vice versa. □

We are now equipped to prove NP-Hardness of CSTMINIMIZATION and follow a reduction similar to that described in proof of Theorem 2 and the reduction from King-Maker graphs, helps us ensure that (i) there exists a vertex cover of smallest size which contains the king, and (ii) every vertex of the graph (except the king) appears in nodes of layer 2 of $\mathcal{C}(\mathrm{ST})$. Also, we append the simplicial complex (associated to) G by introducing a new vertex and extend G through insertion of triangles.

Theorem 4 *CSTMINIMIZATION is NP-Complete.*

Proof It is clear that CSTMINIMIZATION is in NP. We will now show that it is NP-Hard as well. Let G be a King-Maker graph on vertex set V and edge set E, and θ be a labeling of the vertices from 1 to $|V|$. We define K_θ to be the 1-dimensional simplicial complex associated to G. We then introduce a new vertex $\mathtt{v}$ in K_θ with label $|V|+1$ and form maximal triangles, each consisting of $\mathtt{v}$ together with a maximal edge of K_θ. We denote this appended K_θ by K_θ^+. Thus, we still have the same number of maximal simplices but the dimension has increased by 1 and $\mathcal{C}(\mathrm{ST}(K_\theta^+)$ is of height 3 (see Figure 14 for a demonstration). The rest of the proof will focus on proving the following claim: G has a vertex cover of size at most α if and only if there exists a permutation π^+ of the vertices of K_θ^+ such that $|\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta})|$ is at most $2|V|+|E|+\alpha$.

Consider a vertex cover V' of G of size at most α. We construct a permutation π of the labels of the vertices of K_θ so that all vertices in V' appear before any other vertex in V. Further, we assume that the king is in V'. This is no loss of generality since, if a vertex cover excludes the king,

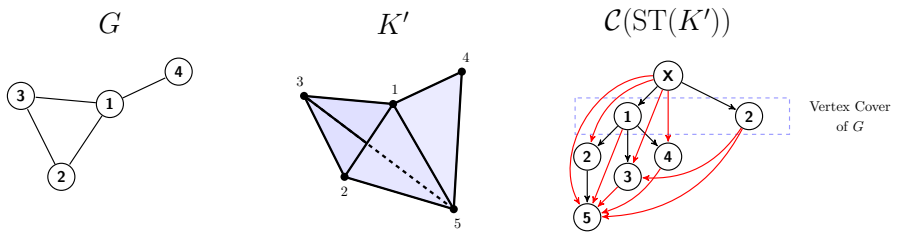

Fig. 14: Demonstration of the construction of K' through an example.

we can always find another vertex cover of the same size which includes the king. Further, we set the label of the king under π to 1. In $\mathcal{C}(\mathrm{ST}(K_{\pi\circ\theta}))$, we distinguish three layers. Layer 1 consists of the root, the vertices of V' appear in layer 2, and all vertices but the king appear in layer 3. There are $|V'|$ edges from layer 1 to layer 2, $|E|$ edges from layer 2 to layer 3 and $|V| - |V'|$ edges from layer 1 to layer 3. Now, we extend π to construct a permutation π^+ of labels of the vertices of K^+_θ by setting $\pi(|V|+1) = |V|+1$ and keeping the same labels as in π for the remaining vertices. We can see $\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))$ as adding one new node and some edges to $\mathcal{C}(\mathrm{ST}(K_{\pi\circ\theta}))$. Node with label $|V|+1$ is put on layer 4 and we introduce an edge from all vertices in layers 1, 2 and 3 to this node in layer 4. Thus we have $|\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))| = |\mathcal{C}(\mathrm{ST}(K_{\pi\circ\theta}))| + 1 + |V'| + (|V| - 1) = 2|V| + |E| + |V'| \leq 2|V| + |E| + \alpha$.

To prove the other direction of the equivalence, we claim that suppose the vertices of K^+_θ can be labeled under a permutation π^+ such that the size of $\mathcal{C}(\mathrm{ST})$ is at most $2|V| + |E| + \alpha$ then, we can find a vertex cover of G of size at most α. In order to find this vertex cover, we observe that the structure of an optimal $\mathcal{C}(\mathrm{ST})$ under π^+ will have only one node in layer 4 and this will contain the vertex $\mathtt{v}$, i.e., the new vertex introduced in K_θ with label $|V|+1$. This is formally stated below as Claim 1. Thus, the structure of $\mathcal{C}(\mathrm{ST})$ is the one we had previously encountered and can extract the vertex cover of G by looking at the nodes on layer 2 (layer 2 can have at most α nodes since we will observe later that in an optimal $\mathcal{C}(\mathrm{ST})$, the king will appear with label 1 under π^+). We will now see that under any permutation if the label of $\mathtt{v}$ is not in the last position then, we can find a new permutation which leads to $\mathcal{C}(\mathrm{ST})$ of smaller size with the label of $\mathtt{v}$ being at the last position.

Claim 1 *Given a permutation π^+ such that $\pi^+(|V|+1) \neq |V|+1$, we can construct a new permutation ρ^+ which is obtained by moving $|V|+1$ to the last position such that:*

$$|\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))| \geq |\mathcal{C}(\mathrm{ST}(K^+_{\rho^+\circ\theta}))|,$$

where we see π^+ and ρ^+ as strings of length $|V|+1$ obtained by their action on $1, 2, \ldots, |V|+1$, and by 'moving' we mean to remove an element from its current position and insert it in the new position.

Thus, if we have a permutation π^+ such that $|\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))| < 2|V|+|E|+\alpha$, we may assume that $\pi^+(|V|+1) = |V|+1$ and we can select the vertex cover of G by considering the vertices on layer 2 of $\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))$. Next, we argue that in π^+, we may assume $\pi^+(1) = 1$ (i.e., the king gets label 1 under π^+), since otherwise, we can always move it to the first position and it does not affect the size of $\mathcal{C}(\mathrm{ST})$. It is now easy to see that the size of layer 2 is less than α as the king is on layer 2.

We know from Lemma 9 that the vertex cover problem for King-Maker graphs is NP-Hard, and this implies that CStMinimization is also NP-Hard. Thus, we have that CStMinimization is NP-Complete. □

Proof of Claim 1 Let $\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta})) \setminus \mathtt{v}$ denote the graph obtained if we delete all the nodes (and all entering or exiting edges incident on these nodes) containing the label of $\mathtt{v}$ in $\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))$. Note that $\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta})) \setminus \mathtt{v}$ is exactly the same as $\mathcal{C}(\mathrm{ST}(K_{\pi\circ\theta}))$ whose size was invariant over π (because K is a 1-dimensional simplicial complex). Therefore, it suffices to show that the sum of the number of edges entering or leaving all nodes containing label of $\mathtt{v}$ in $\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))$ is at least the sum of the number of edges entering or leaving all nodes containing the label of $\mathtt{v}$ in $\mathcal{C}(\mathrm{ST}(K^+_{\rho^+\circ\theta}))$. Let L (resp., R) denote the set of all vertices in G whose label under $\pi\circ\theta$ appear before (resp., after) $\pi^+(\theta(\mathtt{v}))$. Let Δ be the number of edges between L and R in G. Then, we claim that $|\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))| - |\mathcal{C}(\mathrm{ST}(K^+_{\rho^+\circ\theta}))| = \Delta \geq 0$. To see why the claim is true, let us try to analyze the position of the nodes containing label of $\mathtt{v}$ in $\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))$. Write L_v for the label of vertex $\mathtt{v}$, i.e., $L_v = \pi^+(|V|+1)$. If there is an edge between two nodes in R then, we will have a node with label L_v in layer 2. If there is an edge between two nodes in L then, we will have a node with label L_v in layer 4. Finally, if there is an edge between a node in R and a node in L then, we will have a node with label L_v in layer 3. By moving the position of $\mathtt{v}$ to the last position, we are getting rid of the appearance of nodes with label L_v from layer 2 and 3. Now, the king is either in L or in R, and in both cases, we know that there are edges from the king to every other node in $L \cup R$. Therefore, the disappearance of nodes in layer 3, reduces by Δ, the number of edges because all edges between nodes in L in layer 2 and node containing label L_v in layer 4, and all edges between nodes in R in layer 2 and node containing label L_v in layer 4 were already existing in $\mathcal{C}(\mathrm{ST}(K^+_{\pi^+\circ\theta}))$. □

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 4 and Lemma 4 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.

Marc Glisse and Sivaprasad in private communication let us know that they implemented SAL for Data Set 1 on some values of r, and then performed insertion and removal of random simplices, and contracted randomly chosen edges. They made the following observations:

- In all their experiments ($n \approx 10000, r \in [0.2, 0.5]$), size of 1-SAL was significantly smaller than size of ST, and in most cases ST would run out of memory.
- They found that 1-SAL outperformed 0-SAL in low dimensions. However, 0-SAL performed better than 1-SAL in higher dimensions.
- They modified 1-SAL by storing fewer edges and this was noted to save a factor of two in size (in dimension 30), over the 1-SAL proposed in this paper.

Therefore, it would be worth exploring for which class of simplicial complexes, $i-$SAL is the best data structure in the SAL family (for every $i \in \mathbb{N}$).

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 [11] and [26], and extended to acyclic automata in [14] which would be of particular interest to us. Interestingly, it appears that in all of the works above, the finiteness of the language plays no special role and, for the specific case of SA, results may be made sharper.

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 [21] and since our language is finite, $k-$minimization [6] and cover automata [13] might give efficient approximate data structures by hyperminimizing SA. We motivate this with the help of building complexes from a random sample of a large data set. By sampling we are bound to lose information and, instead of taking a random sampling we can look at constructing hyperminimized SA over the entire data set dynamically. It will be interesting to know if the power of randomness can overcome 'smart' approximations.

Theorem 3 can be generalized to give the following hardness result: Given a word w on alphabet set Σ and lexicographic ordering θ on Σ, let $\mathcal{L}_\theta$ be an operation which removes all duplicate letters in the word and rearranges the

letters of the word in lexicographic order given by θ. Given a language L on alphabet set Σ, we define $\mathcal{L}_\theta(L) = \{\mathcal{L}_\theta(w) \mid w \in L\}$. Therefore with these definitions, we give the following general result:

Theorem 5 *Given a finite language L (explicitly through the words in L) on alphabet set Σ and an integer x, it is NP-Hard to decide if there exists an ordering θ on Σ such that the size of the smallest DFA recognizing $\mathcal{L}_\theta(L)$ is less than x.*

Theorem 4 and 5 provide a new dimension to the hardness results obtained by Comer and Sethi in [12]. It would be worth exploring this direction further. Also, it would be interesting to find approximation algorithms for MsaMinimization.

Performance of i-SAL depends on the value of Γ_i - are there any interesting bounds on Γ_i for some subset of nice simplicial complexes of $\mathcal{K}(n, k, d, m)$? 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.

9 Acknowledgement

We would like to thank Eylon Yogev for helping with carrying out some experiments. We would like to thank Rajesh Chitnis for pointing out a short proof of Lemma 8. We would like to thank François Godi for pointing out a mistake in the analysis of the cost of the edge contraction operation for the Simplex Array List as it appeared in [7]. We would like to thank Marc Glisse for several comments on earlier versions of this paper and also for pointing out a tightening of the cost of the edge contraction operation for the Simplex Array List. We would like to thank Marc Glisse and Sivaprasad S. for implementing SAL and sharing their results with us. Finally, we would like to thank Dorian Mazauric for pointing out Theorem 2.

References

1. A.W. Appel and G.J. Jacobson: The world's fastest scrabble program, *In Communications of the ACM*, Vol 31, 1988.
2. 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.
3. A. Andersson and S. Nilsson: Improved Behaviour of Tries by Adaptive Branching, *In Information Processing Letters*, Vol 46, pages 295–300, 1993.
4. 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.
5. 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.
6. A. Badr, V. Geffert, and I. Shipman: Hyper-minimizing minimized deterministic finite state automata. *In RAIRO Theoretical Informatics and Applications*, 43(1), pages 69–94, 2009.

7. J-D. Boissonnat, Karthik C. S., and S. Tavenas: Building Efficient and Compact Data Structures for Simplicial Complexes, *In Symposium on Computational Geometry*, pages 642–656, 2015.
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, *In Theoretical Computer Science, 617,* pages 28–44, 2016.
10. J. L. Bentley and R. Sedgewick: Fast algorithms for sorting and searching strings. *In Proceedings of the eighth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 360−369, 1997.
11. R. Carrasco and M. Forcada: Incremental construction and maintenance of minimal finite-state automata. *In Computational Linguistics*, Volume 28, 2002.
12. D. Comer and R. Sethi: Complexity of Trie Index Construction, *In Proceedings of Foundations of Computer Science*, pages 197-207, 1976.
13. C. Câmpeanu, N. Sântean, and S. Yu: Minimal cover-automata for finite languages. *In Theoretical Computer Science* 267(1–2), pages 3–16, 2001.
14. J. Daciuk, S. Mihov, B. Watson, and R. Watson: Incremental construction of minimal acyclic finite-state automata. *In Computational Linguistics*, Volume 26, pages 3–16, 2000.
15. D. Eppstein, M. Löffler, and D. Strash: Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time, *In International Symposium on Algorithms and Computation (1)*, pages 403-414, 2010.
16. M. Golumbic: Algorithmic Graph Theory and Perfect Graphs. *In Academic Press*, 2004.
17. M. R. Garey and D. S. Johnson: Computers and Intractability: a guide to the theory of NP-completeness. *W. H. Freeman publishers*, 1979.
18. M. Grohe, S. Kreutzer, and S. Siebertz: Characterisations of Nowhere Dense Graphs, *In IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 21-40, 2013.
19. J. Hopcroft: An n log n algorithm for minimizing states in a finite automaton. *In Theory of machines and computations*, pages 189–196, 1971.
20. R. M. Karp: Reducibility Among Combinatorial Problems. *In R. E. Miller and J. W. Thatcher (editors). Complexity of Computer Computations*, pages 85–103, 1972.
21. A. Maletti: Notes on hyper-minimization. *In Proceedings 13th International Conference Automata and Formal Languages*, pages 34–49, 2011.
22. E. Maia, N. Moreira, and Rogrio Reis: Incomplete Transition Complexity of Some Basic Operations, *In International Conference on Current Trends in Theory and Practice of Computer Science*, pages 319–331, 2013.
23. A. Nerode: Linear Automaton Transformations, *In Proceedings of the American Mathematical Society*, Volume 9, pages 541−544, 1958.
24. D. Revuz: Minimisation of acyclic deterministic automata in linear time, *In Theoretical Computer Science, Volume 92, Issue 1*, pages 181–189, 1992.
25. K. Sgarbas, N. Fakotakis, and G. Kokkinakis: Optimal insertion in deterministic DAWGs. *In Theoretical Computer Science*, pages 103–117, 2003.
26. J. Teuhola and T. Raita: Text compression using prediction, *In Proceedings of ACM Conference on Research and Development in Information Retrieval*, pages 97–102, 1986.
27. D. Yellin: Algorithms for Subset Testing and Finding Maximal Sets, *In SODA*, pages 386–392, 1992.

A Adapted Hopcroft's Algorithm

We precisely describe here Hopcroft's algorithm adapted to the compression of ST to provide as output $\mathcal{C}(\mathrm{ST})$. The idea is to write the Simplex Tree as the Simplex Automaton: each vertex of the tree becomes a state in the automaton. Then, one just needs to reduce the Simplex Automaton using Hopcroft's algorithm and finally obtain the compressed Simplex Tree by partitioning the states of the Minimal Simplex Automaton. We see in Figure 4, the compressed Simplex Tree of the simplicial complex described in Figure 1.

Algorithm 1 Hopcroft's Algorithm for compression of Simplex Tree

Input: A Simplex Tree $\mathcal{T}$. Let L be the set of leaves, N be the set of internal nodes and V be the set of the vertices of the simplicial complex.
Output: A compressed Simplex Tree $\mathcal{T}'$.

$\mathcal{S} \leftarrow \emptyset$
$\mathcal{P} \leftarrow \{L, N\}$
for $v \in V$ **do**
  Add (L, v) in $\mathcal{S}$.
end for
while $\mathcal{S} \neq \emptyset$ **do**
  Pop one element (C, v) in $\mathcal{S}$.
  for each $B \in \mathcal{P}$ such that there exists n_1 and n_2 in B such that there is an edge in $\mathcal{T}$ from n_1 to a node of C labelled by v, and it is not the case for n_2 **do**
    $B' \leftarrow \{n \in B | \text{there is an edge from } n \text{ to a node in } C \text{ labelled by } v\}$
    $B'' \leftarrow \{n \in B | \text{there is no edge from } n \text{ to a node in } C \text{ labelled by } v\}$
    Replace B by B' and B'' in $\mathcal{P}$.
    for $w \in V$ **do**
      if $(B, w) \in \mathcal{S}$ **then**
        Replace (B, w) by (B', w) and (B'', w) in $\mathcal{S}$.
      else
        $D \leftarrow \begin{cases} B' \text{ if } |B'| \leq |B''| \\ B'' \text{ otherwise.} \end{cases}$
        Add (D, w) in $\mathcal{S}$.
      end if
    end for
  end for
end while
Now we just have to build the compressed Simplex Tree $\mathcal{T}'$.
Let S be the part in $\mathcal{P}$ which contains the root of $\mathcal{T}$.
$\mathcal{T}' \leftarrow$ initial state: $(S, \star)$ labeled $\star$.
$\mathcal{V} \leftarrow (S, \star)$
while $\mathcal{V} \neq \emptyset$ **do**
  Pop an element (B, w) in $\mathcal{V}$.
  for $v \in V$ **do**
    Let b be an element in B. * It will be a representative of B.

Let a be (if it exists) the node in $\mathcal{T}$ reached from b by reading v.
Let A be the part in $\mathcal{P}$ which contains a.
if (A, v) is not already in $\mathcal{T}'$ **then**
Add a vertex (A, v) in $\mathcal{T}'$.
Push (A, v) in $\mathcal{V}$.
end if
Add an edge from (B, w) to (A, v) in $\mathcal{T}'$.
end for
end while

B Operations on the Maximal Simplex Tree

We provide below a list of basic operations on the MxST. In the following subsections we denote by T_{MxST} the maximum time taken to search for particular children of any node in MxST ($T_{\mathrm{MxST}} = \mathcal{O}(\log n)$ for red–black trees).

B.1 Identifying the Maximal Cofaces of a Simplex

As seen in section 4, a simplex $\sigma \in K$ is implicitly stored in MxST(K) as a face of its (at most k) maximal cofaces. Identifying the maximal cofaces of a simplex means to find the leaves of MxST(K) that contain the maximal cofaces of σ and also to find, on each path from the root to such a leaf, the location of the vertices of σ.

We formalize this by first defining a bijection f from the set of all maximal simplices of K to the set of all leaves in MxST(K) and also define a function g that maps every simplex $\sigma \in K$ to the set of all maximal simplices in K which contain σ (thus if $f(\sigma) = \emptyset$ then $\sigma \notin K$). For simplicity, we will denote $f(g(\sigma))$ by $L(\sigma)$ and $|L(\sigma)|$ by k_σ from now on. Since identifying a simplex σ reduces to traversing MxST(K), this operation costs $\mathcal{O}(kdT_{\mathrm{MxST}}) = \mathcal{O}(kd \log n)$.

B.2 Insertion

Let σ be a simplex not in K and let K' be the complex obtained by adding σ and its subsimplices to K. Observe that σ is necessarily maximal in K' and write d_σ for the dimension of σ. We describe how to insert σ in MxST. We first check if there exists maximal simplices in MxST(K) that are contained in σ. This can be done in $\mathcal{O}(kd_\sigma T_{\mathrm{MxST}})$ time, by looking at the MxST truncated to depth d_σ. If such simplices exist, we will need to delete them before inserting σ, which takes time at most $\mathcal{O}(kd_\sigma T_{\mathrm{MxST}})$ (see analysis of step 3 of Algorithm 2). Then, we insert σ in MxST. This takes at most $\mathcal{O}(d_\sigma T_{\mathrm{MxST}})$ time, which is significantly better than the time taken for the Simplex Tree, which needs $\mathcal{O}(2^{d_\sigma} T_{\mathrm{MxST}})$ time. We conclude that the time for inserting σ is $\mathcal{O}(kd_\sigma T_{\mathrm{MxST}}) = \mathcal{O}(kd_\sigma \log n)$.

B.3 Removing a Face

Given a simplex σ, we have to remove it (and its cofaces) from the MxST. We can perform the operation of removing simplices (the simplex and its cofaces) as described in Algorithm 2.

Algorithm 2 Removing simplices in Maximal Simplex Tree

Input: A Maximal Simplex Tree and a simplex σ.
Output: A Maximal Simplex Tree.
1: Compute $L(\sigma)$.
2: **for** each $\Gamma \in L(\sigma)$ **do**
3: Remove the branch ending at Γ.
4: for every vertex v in σ insert $\Gamma \setminus \{v\}$.
5: **end for**

Step 1 was shown earlier to take $\mathcal{O}(kdT_{\mathrm{MxST}})$ time. Since Γ is maximal, it is associated to a leaf in the tree. Let $P(\Gamma)$ be the path from the root to the leaf associated to σ. For removing a maximal simplex Γ, one needs to locate on $P(\Gamma)$, the last node w such that the out–degree for the edges is strictly more than one (it corresponds to the last node which is shared with another maximal simplex). Then, one just has to delete the edge on the corresponding path going from w. So removing one maximal simplex (at step 3) takes time $\mathcal{O}(dT_{\mathrm{MxST}})$ (since we might have to potentially delete up to d nodes). At step 4, d_σ facets of Γ need to be inserted and each insertion was shown earlier to be doable in time $\mathcal{O}(T_{\mathrm{MxST}}d_\Gamma))$ (since we do not have to check if there exists maximal simplices in $\mathrm{MxST}(K)$ that are contained in facets Γ). The total algorithm costs time $\mathcal{O}(\mathrm{k}_\sigma dT_{\mathrm{MxST}}+\mathrm{k}_\sigma T_{\mathrm{MxST}}d_\Gamma) = \mathcal{O}(kd\log(n))$.

B.4 Elementary Collapse

An elementary collapse consists of removing both simplices of a free pair. It preserves the homotopy type of the complex. We break down the computation of an elementary collapse into 3 steps which is described in Algorithm 3.

Algorithm 3 Elementary collapse in Maximal Simplex Tree

Input: A Maximal Simplex Tree and a pair of simplices (τ, σ).
Output: A Maximal Simplex Tree.
1: Check if (τ, σ) is a free pair.
2: Delete τ.
3: Insert the facets of τ which are different from σ, and are now maximal.

It easily follows from the above results that Step 1 takes $\mathcal{O}(kd\log(n))$ time. As for the removing operation, step 2 is feasible in time $\mathcal{O}(dT_{\mathrm{MxST}})$ (see analysis of step 3 of Algorithm 2). For step 3, we have d_τ facets to check if they are now maximal and then insert the ones that are indeed maximal. This can be done in time $\mathcal{O}(kdd_\sigma\log(n))$.

B.5 Edge Contraction

Edge contraction is another operation on simplicial complexes that preserves the homotopy type under certain conditions and which can be implemented on the MxST using the above operations. We break down the computation of an edge contraction into 3 steps which is described in Algorithm 4.

Steps 1 and 3 can be done in time $\mathcal{O}(kd\log n)$ (follows from our previous results). However, to perform step 2, we need to remove the simplices which were previously maximal,

Algorithm 4 Edge contraction in Maximal Simplex Tree

Input: A Maximal Simplex Tree and a pair of vertices (u, v).
Output: A Maximal Simplex Tree.
1: Replace the label u by the label v each time u appears in the MxST.
2: Store the list L of all the maximal simplices (lexicographically).
3: Build the MxST from L.

but are no longer maximal after the edge contraction. This can be done in time $\mathcal{O}(kd \cdot k')$ [28], where k' is the number of maximal simplices after the edge contraction. So an edge contraction can be performed in time $\mathcal{O}(kd \log n + kdk') = \mathcal{O}(kd(k + \log n))$. Finally, we note that the above algorithm is a multiplicative factor of $k/\log n$ in the worst case away from the upperbound for MxST. For any k, d, let us consider the simplicial complex in Example 6.

Example 6 Let $K \in \mathcal{K}(kd/4 + k/2 + d + 1, k, d, m)$ be defined by the union of the sets of maximal simplices given below:

1. $\{1, 2, \ldots, d, d + i | 1 \leq i \leq k/2\}$.
2. $\{i \cdot d + 1 + k/2, i \cdot d + 2 + k/2, \ldots, i \cdot d + k/2 + d \mid 1 \leq i \leq k/4\}$.
3. $\{i \cdot d + 1 + k/2, i \cdot d + 2 + k/2, \ldots, i \cdot d + k/2 + d - 1, kd/4 + k/2 + d + 1 \mid 1 \leq i \leq k/4\}$.
4. $\{1, kd/4 + k/2 + d + 1\}$.

If the vertex $kd/4 + k/2 + d + 1$ is contracted to 1 then, the new simplicial complex is generated by the union of the sets of maximal simplices given below:

1. $\{1, 2, \ldots, d, d + i | 1 \leq i \leq k/2\}$.
2. $\{i \cdot d + 1 + k/2, i \cdot d + 2 + k/2, \ldots, i \cdot d + k/2 + d \mid 1 \leq i \leq k/4\}$.
3. $\{1, i \cdot d + 1 + k/2, i \cdot d + 2 + k/2, \ldots, i \cdot d + k/2 + d - 1 \mid 1 \leq i \leq k/4\}$.

The new MxST has at least $k(d-1)/4$ new nodes (of size $\log n$) that need to be added. Thus, we have a lower bound of $\Omega(kd \log n)$ on the operation of edge contraction for MxST.