6-1-2012

Constructing Simplicial Complexes over Topological Spaces

Milka N. Doktorova
Dartmouth College

CONSTRUCTING SIMPLICIAL COMPLEXES OVER TOPOLOGICAL SPACES

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Master of Science

in

Computer Science

by

Milka Doktorova

DARTMOUTH COLLEGE

Hanover, New Hampshire

June 2012

Examining Committee:

Afra Zomorodian, Ph.D. (advisor)

Gevorg Grigoryan, Ph.D.

Devin Balkcom, Ph.D.

Dartmouth Computer Science Technical Report TR2012-721

Abstract

The first step in topological data analysis is often the construction of a simplicial complex. This complex approximates the lost topology of a sampled point set. Current techniques often assume that the input is embedded in a metric – often Euclidean – space, and make significant use of the underlying geometry for efficient computation. Consequently, these techniques do not extend to non-Euclidean or non-metric spaces. In this thesis, we present an oracle-based framework for constructing simplicial complexes over arbitrary topological spaces. The framework consists of an oracle and an algorithm that builds the simplicial complex by making calls to the oracle. We compare different algorithmic approaches for the construction, as well as alternate ways of representing the simplicial complex in the computation. Finally, we demonstrate the utility of our framework as a tool for approaching problems of diverse nature by presenting three applications: to multiword search in Google, to the computational analysis of a language and to the study of protein structure.

Acknowledgements

I would like to thank Afra Zomorodian, Gevorg Grigoryan and Devin Balkcom for being on my thesis committee. Afra introduced me to the idea for this project and thanks to his invaluable advice and support I was able to start it and bring it to the form in which it is presented here. Gevorg taught me a lot about proteins and computational biology in general, and I am very grateful for his help with the protein application. Devin introduced me to some interesting problems in robotics which helped me expand my understanding on covers and space representations, and consequently made me think of other possible applications of this work.

I want to extend a special note of gratitude to the Dartmouth Computer Science department for their understanding and support in my transition to the Master's program.

I want to thank my family and friends for their unconditional support all throughout.

This research has been partially supported by ONR N 00014-08-1-0908 and NSF CAREER CCF-0845716. Automated queries to Google were performed via license #731826940 from University Research Program for Google Search.

Contents

Chapter 1

Introduction

Graphs are a combinatorial model for capturing pairwise relationships between objects. In a graph, the nodes represent the objects, and an edge between two nodes indicates that the corresponding pair of objects has this relationship. Similarly, *hypergraphs* can capture multi-way relationships as their edges can connect any number of nodes. Here we are interested in combinatorial structures that capture multi-way relationships between objects, and come out of algebra. These structures, called *simplicial complexes*, have been popular in a number of disciplines, such as computational geometry, computer graphics, and lately, topological data analysis. Generally, the complexes are built over metric, often Euclidean, spaces using algorithms that make fundamental use of the geometry of the space. However, we are often faced with the task of analyzing point sets that come from non-metric spaces, where none of the current techniques may be used. The focus of this thesis then, is constructing simplicial complexes over arbitrary topological spaces.

1.1 Motivation

This work is motivated naturally by two classes of problems. First, we use simplicial complexes as combinatorial models that represent the lost topology of a point set.

As such, construction of appropriate complexes is the first step in *topological data analysis*: the process of obtaining the global connectivity of a space from local information [5, 26]. Suppose, for instance, that S is a finite set of points in $\mathbb{R}^n$, $n \gg 2$, drawn from the surface of a 2-dimensional sphere. Given S in this high n-dimensional space, we use topological analysis to try and recover the *underlying space* from which S was sampled, namely the 2-dimensional sphere. The process involves building a simplicial complex that approximates the structure of the underlying space, then studying its topology. The applicability and efficiency of the construction depend on the *embedding* space A, i.e. the space where the points reside, which in our example is $\mathbb{R}^n$. Existing algorithms assume that A is a metric space and take advantage of its structure. However, due to this assumption the algorithms cannot be extended to the analysis of point sets in spaces whose structure may be unknown. We address this problem by presenting a unified framework for constructing simplicial complexes without making any assumptions about the embedding space. As a result, we have a tool for analyzing datasets that may be heterogeneous, high-dimensional, and non-metric.

In addition, as discussed earlier we may utilize a simplicial complex to model relationships between entities. We illustrate this concept here with a concrete application to which we return at the end of the thesis. Suppose we have a list of ingredients $L = \{$cheese, carrots, milk, eggs, butter, red pepper$\}$ and want to find a recipe online that includes some or all of them. Searching for all ingredients in a single query in Google, gives 14.5 million hits, while restricting the search to only a particular cuisine such as Indian, gives 0 hits. Our goal is to find the combinations of the query terms that give a small, but non-zero number of results. Since the number of possible combinations grows exponentially with the size of the list, we also want to do this efficiently. We address this multiword search problem by

constructing a simplicial complex on the elements in L as outlined in Section 3.1. As a result, we get 6 maximal combinations of the ingredients after a total of 26 queries, with each combination containing between 1 and 3 hits. Furthermore, the number of queries made by our algorithm in general is significantly lower than the total possible number of queries, as illustrated in Figure 1.1. A more rigorous related analysis is presented in Section 5.1.

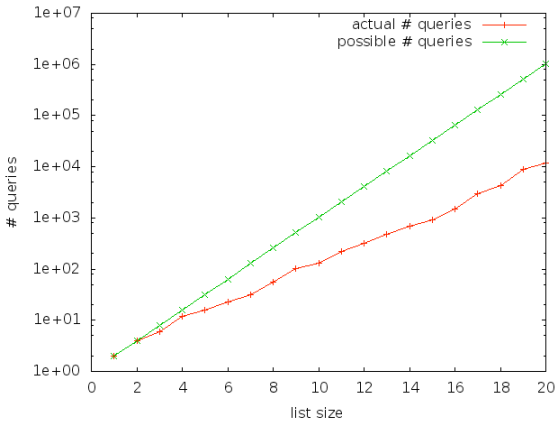


Figure 1.1: Log graph of the *possible* (2^l) number of queries and the *actual* number of queries made to find all combinations of the items in a list of size l that result in nonzero hits.

1.2 Prior Work

We divide prior techniques for the construction of simplicial complexes roughly into two groups: constructing nerves of covers and building clique complexes.

Nerves are simplicial complexes built on a given *cover* of the input data points. Intuitively, a cover is a collection of *cover sets*, or different, possibly overlapping subsets of the points. The algorithms for nerve construction differ in the way in which they build the cover; the nerve of the cover then is just a simplicial complex on the cover sets. A classic algebraic construction is the Čech complex [14] in which the cover is the union of ϵ-balls centered at the input data points. Building the Čech complex however, is infeasible in practice due to the involved computational complexity. The Delaunay [7] and Alpha [9, 10] complexes, on the other hand, are instances of structures that take advantage of the geometry of the embedding space. They are based on the Voronoi diagram and a set of restricted Voronoi regions, respectively. One major limitation of all these methods in the context of our work,

however, is that they are defined only for metric spaces.

An alternate approach for the construction is building the clique complex of a graph. This approach first captures the geometry of the space in a graph, then adds higher-dimensional simplices using an algebraic procedure. In a graph, a *clique* is a set of nodes that are pairwise connected, and a *maximal clique* is a clique that cannot be made any larger. The clique or flag complex [18] has maximal cliques as its maximal simplices. There are currently two popular instances of this approach. The Vietoris-Rips (VR) complex builds a neighborhood graph on the input point set that has edges with length less than some parameter ϵ [23]. In contrast, the weak witness complex builds a graph on a subset of the input based on a relaxation of the Delauany triangulation [8]. Note that if we have a graph encoding the pairwise relationships between data, the construction of the clique complex of this graph assumes that if n data points are all pairwise related, then all n points are related. However, this assumption is too restrictive and may not always hold, as in the problem of multiword search.

Examples of other models that approximate the structure of the underlying space in the first stage of topological data analysis include the flow complex [12] and the cubical complex [15] but they are again, defined only for metric spaces.

In addition to constructing simplicial complexes, we are also interested in the efficient computational representation of the structures. The problem of simplifying the complexes while retaining their key properties has been approached in different ways. Basic simplification operations include vertex removal [22], vertex clustering [21], and edge contraction. The latter has been used by both Zomorodian [25] and Attali et al. [2] in recent studies. While in [25] a clique complex is represented with a tidy set that captures the topology of the complex, Attali et al. present a new data structure for the efficient representation of complexes that are "close" to flag complexes. The data structure encodes a simplicial complex K by its underlying graph G together

with a set of *blockers*: the inclusion-minimal simplices in the set difference $Flag(G)\backslash K$ where $Flag(G)$ is the full flag complex on G. In comparison, two of our algorithms construct and store the complex by keeping only its inclusion-maximal, or just maximal, simplices. Note that the problem of enumerating all maximal simplices in clique complexes in particular, is closely related to the problem of enumerating all maximal cliques in a graph which is a combinatorics problem [6]. For an overview of different variants of combinatorial algorithms, see [17].

1.3 Contributions

In this thesis, we provide algorithms for constructing a simplicial complex over an arbitrary topological space, the most general form of a space that still retains a notion of connectivity. The particular contributions of our work appear as follows:

- We introduce a general oracle-based framework for describing the space in Section 3.1.
- We present four algorithms that build a simplicial complex in two different representations in Section 3.2.
- We implement all algorithms and analyze their performance through a number of experiments in Section 4.
- We give three applications of our work in Section 5, returning to our motivating search problem.

Chapter 2

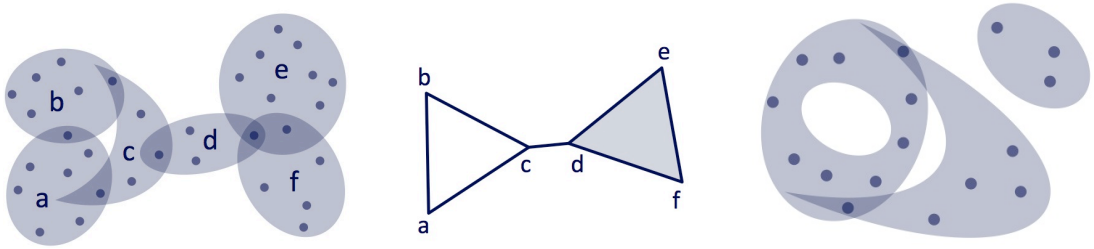
Background

In this chapter we review the properties of simplicial complexes and define some terminology used throughout the thesis. In addition, we discuss two different representations of the combinatorial structures, as they play a role in the efficiency of the algorithms described later. We conclude by defining *cover* and *nerve*.

2.1 Simplicial Complex

2.1.1 Definition

A *simplicial complex* K is a set of finite sets with the property that if $\sigma \in K$, any subset $\tau \subseteq \sigma$ is also in K. Each element σ in K with cardinality $|\sigma| = k + 1$ is called a *simplex* of dimension k, or a *k-simplex*. All simplices of dimension $k \leq d$ form the *d-skeleton* of the complex. Simplicial complexes may be realized geometrically: for instance, a k-simplex is a vertex, an edge, a triangle or a tetrahedron for $0 \leq k \leq 3$, respectively [7]. The 1-skeleton of the complex therefore, may be viewed as a graph formed by its 0- and 1-dimensional simplices. The dimension of the complex is the dimension of the highest-dimensional simplex in the complex, i.e. $\dim(K) = \max_{\sigma \in K} \dim(\sigma)$. The simplicial complex K' in Figure 2.1(b) consists of six



(a) 33 points and a cover of 6 sets with labels for each cover set

(b) a simplicial complex K': the nerve of the cover in (a)

(c) 17 points and a cover of 3 sets

Figure 2.1: Examples of covers and nerve.

0-simplices, seven 1-simplices, and one 2-simplex, and has dimension 2. For ease of notation we denote simplices $\{u, v\}$ and $\{u, v, w\}$ with $\{uv\}$ and $\{uvw\}$ respectively:

$$\begin{aligned} K' = \{ \quad & \emptyset, \\ & \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \\ & \{ab\}, \{bc\}, \{ac\}, \{cd\}, \{de\}, \{ef\}, \{df\}, \\ & \{def\} \quad \}. \end{aligned}$$

For every subset τ of a simplex σ, we say that τ is a *face* of σ, its *coface*. For example, the vertex $\{d\}$ and the edge $\{ef\}$ are two faces of the triangle $\{def\}$, and the edge $\{ac\}$ is a coface of the vertices $\{a\}$ and $\{c\}$. The $(n-1)$-dimensional faces of an n-dimensional simplex are also called *facets* of the simplex.

A *poset*, or a partially ordered set, is a set equipped with a *partial order*, i.e. a relation that is reflexive, antisymmetric and transitive [11]. A simplicial complex may be viewed as a poset in which the simplices are the elements of the set, and the face relation between them is the partial order. A simplex is *maximal* if it has no proper coface in K. In Figure 2.1(b) the maximal simplices are $\{ab\}$, $\{bc\}$, $\{ac\}$,$\{cd\}$ and $\{def\}$. The maximal simplices of the complex are thus the maximal elements of the poset. As such, they form an *antichain* A in the poset, that is, a set of pairwise incompatible elements: if $u, w \in A$, then $u \not\subseteq w$ and $w \not\subseteq u$.

2.1.2 Representation

We can represent a simplicial complex by all of its simplices, in its *full representation*, or only by its maximal simplices, in its *maximal representation*. Each representation offers certain advantages and disadvantages. If we store all simplices of the complex, checking for the existence of a particular simplex takes $O(1)$ time but the complex can be very large in size in which case storage may become infeasible. Keeping only the maximal simplices on the other hand, significantly reduces the size of the complex as discussed later, but simplex lookups become more expensive. Moreover, if we want to store some additional information, i.e. a *weight*, for each simplex, we can do this directly if all simplices are present in the complex, while with maximal representation we can store weights only for the maximal simplices.

Note that since simplicial complexes are closed under inclusion, we can switch between maximal and full representation in time linear in the size of the complex: if we have only the maximal simplices, we can enumerate and find all simplices, and if we have all simplices, we can mark and keep only the maximal ones.

2.2 Covers and Nerve

Let S be a set of points in some topological space A. A *cover* U of S is:

$$U = \{\mathbb{U}_i\}_{i \in I}, \qquad \mathbb{U}_i \subseteq A,$$

where I is an indexing set and $S \subseteq \cup_i \mathbb{U}_i$. The *nerve* N of the cover is a simplicial complex on the sets in the cover, where set intersection is the predicate. Formally, given a cover U, N is defined as:

(1) $\emptyset \in N$, and

(2) If $\cap_{j \in J} \mathbb{U}_j \neq \emptyset$ for $J \subseteq I$, then $J \in N$.

Figure 2.1(a) is an example of a point set and one possible cover on the points, and Figure 2.1(b) illustrates the nerve of the cover.

Not every cover yields a nerve that correctly captures the topology of a space. According to Leray's classical *Nerve Lemma* [4], U is a *good* cover if all $\mathbb{U}_i$ are *contractible* and so are all of their nonempty intersections. A topological space X is contractible if it has trivial reduced homology groups, but a full exposition of homology is outside the scope of this thesis [14]. Intuitively, X is contractible if it has no interesting topology like, for example, a disk or $\mathbb{R}^d$, $d \in \mathbb{Z}^+$. The cover on Figure 2.1(a) is a good cover while that on Figure 2.1(c) is not because the leftmost set is an annulus, and its intersection with the middle set has two pieces.

Chapter 3

Framework

In this chapter, we present the details of our oracle-based framework. We first introduce our modeling approach, then discuss four different algorithms that perform the construction.

3.1 Approach

A predicate $f : A \to \{T, F\}$ is *monotonic* if $f(A) = T$ implies $f(C) = T$ for all $C \subseteq A$. Suppose $B = \{b_i\}_{1 \leq i \leq n}$ is a set of n objects, and $\mathcal{V} = \{v_i\}_{1 \leq i \leq n}$ is a set of n vertices. Let $g : B \to \mathcal{V}$ be a bijective function that maps each object $b_i \in B$ to a vertex $v_j \in \mathcal{V}$ for $i, j \in [1, n]$. We define our problem as follows:

Definition 1. *(Multiword Search Problem) Given a set of n vertices $\mathcal{V}$ and a monotonic oracle $\mathcal{O} : 2^{\mathcal{V}} \to \{T, F\}$, find a subset $W \subseteq \mathcal{V}$ such that W is valid, i.e. $\mathcal{O}(W) = T$, and W has maximal size, i.e. $|W| \geq |W'|$ for all valid subsets $W' \subseteq \mathcal{V}$.*

The oracle in the definition is a predicate that maps subsets of $\mathcal{V}$ to true (T) or false (F). We use it to build a simplicial complex K on the vertices in $\mathcal{V}$. In this context, $\mathcal{O}$ can intuitively be thought of as a black box that determines whether simplices exist or not. A vertex $v_i \in \mathcal{V}$, for example, is a 0-simplex in K if $\mathcal{O}(\{v_i\}) = T$;

two vertices, v_i and v_j form a 1-simplex if $\mathcal{O}(\{v_i, v_j\}) = T$; three vertices, v_i, v_j, v_l form a 2-simplex if $\mathcal{O}(\{v_i, v_j, v_l\}) = T$, and so on. The monotonicity requirement for the oracle ensures that we have a simplicial complex. Note however that, as a consequence of our problem formulation, there is no order on the vertices that the oracle takes as input: if w is an ordered k-tuple of vertices, $w = (v_i)_{i \in I}$ with $k = |I|$, then $\mathcal{O}(w) = \mathcal{O}(\pi(w))$ where $\pi(w)$ is any permutation on w.

Let $\mathcal{V}_i$ be a cover set corresponding to vertex $v_i \in \mathcal{V}$. If $J \subseteq \{1, 2, ..., n\}$, then for any subset $\mathcal{V}_J = \{v_j \mid j \in J\}$, the oracle returns T if the intersection of the corresponding cover sets is nonempty:

$$\mathcal{O}(\mathcal{V}_J) = T \qquad \Leftrightarrow \qquad \bigcap_{j \in J} \mathcal{V}_j \neq \emptyset.$$

Hence, the simplicial complex K that we build is the nerve of the cover $\mathcal{C}$ defined by the sets $\mathcal{V}_i$, i.e. $\mathcal{C} = \{\mathcal{V}_i\}_{1 \leq i \leq n}$.

Each simplex in K corresponds to a valid subset of the vertices. Thus, the subset $W \subseteq \mathcal{V}$ that we want to find should appear as a maximal simplex in the complex. Since the oracle may be very slow, our goal is to construct the simplicial complex by making as few calls to the oracle as possible.

3.2 Algorithms

We now present four algorithms that use the oracle to build a simplicial complex on the set of vertices $\mathcal{V}$.

3.2.1 Tools

First, we introduce three subroutines to which we refer throughout the section. Let $G = (V, E)$ be a graph with a total ordering $<$ on V (an arbitrary ordering would suffice). The following ADD-NODE$(G, v, \mathcal{O})$ subroutine uses the oracle $\mathcal{O}$ to add the

vertex v and its neighborhood to G. Throughout, dot notation denotes access to the elements of G.

ADD-NODE$(G, v, \mathcal{O})$

1 $G.V \leftarrow G.V \cup \{v\}$

2 **foreach** $u \in G.V$, $u \neq v$

3 **do if** $\mathcal{O}(\{u, v\})$

4 **then** $G.E \leftarrow G.E \cup (u, v)$

We define the *lower neighbors* of a vertex $u \in V$ to be all vertices $v \in V$ such that $v < u$ and $(v, u) \in E$. We find them by calling LOWER-NBRS(G, u):

LOWER-NBRS(G, u)

return $\{v \in G.V \mid v < u, (v, u) \in G.E\}$.

Now, suppose σ is a simplex in a simplicial complex K constructed on the vertices V. The lower neighbors of σ are the shared lower neighbors among all of σ's 0-dimensional faces. The corresponding set is retrieved by the SIMPLEX-LOWER-NBRS function:

SIMPLEX-LOWER-NBRS(G, σ)

return $\{v \in G.V \mid v \in \bigcap_{u \in \sigma}$ LOWER-NBRS$(G, u)\}$.

Using these subroutines, we present four algorithms: INCREMENTAL, INDUCTIVE, MAXIMAL-BU (Bottom-Up) and MAXIMAL-TD (Top-Down). All algorithms, except for INDUCTIVE, are incremental in nature as they build the complex gradually by taking vertices one by one. Note that INDUCTIVE and INCREMENTAL are similar to the VR algorithms in [24] and keep the complex in its full representation. MAXIMAL-BU is algorithmically equivalent to INCREMENTAL but stores only the maximal simplices in the complex. MAXIMAL-TD also keeps the complex in its maximal representation but performs the construction in a top-down manner.

3.2.2 Incremental

We begin with INCREMENTAL which is an algorithm based on INCREMENTAL-VR. The procedure constructs a simplicial complex by considering vertices one by one and building a graph G. For each vertex $v \in \mathcal{V}$, we add v to V, find all of v's neighbors and use them to try and form new simplices of the complex K. The construction is thus performed in an incremental manner as follows:

```
INCREMENTAL(V, O)
1  V = E = ∅
2  G = (V, E)
3  foreach vertex v ∈ V
4        do if O({v})
5              then ADD-NODE(G, v, O)
6                   N ← LOWER-NBRS(G, v)
7                   BUILD-INCREMENTAL({v}, O, G, N, K)
8  return K
```

```
BUILD-INCREMENTAL(τ, O, G, N, K)
 1  K ← K ∪ τ
 2  foreach v_n ∈ N:
 3        do σ ← τ ∪ {v_n}
 4           foreach facet f of σ
 5                do if f ∉ K
 6                      then if O(f)
 7                                then K ← K ∪ f
 8           if all facets of σ exist
 9              then if O(σ)
10                      then N ← N ∩ LOWER-NBRS(G, v_n)
11                           BUILD-INCREMENTAL(σ, O, G, N, K)
```

Note that in order to check if σ exists, we first check all of its facets in the for loop on Line 4 in BUILD-INCREMENTAL.

3.2.3 Inductive

Our second algorithm, INDUCTIVE, constructs the simplicial complex by inducting on the dimension of the simplicial complex. It starts by building the 1-skeleton of the complex. Following that, it loops through all simplices of dimension d (for $d \geq 1$) and for every simplex tries to form $(d+1)$-dimensional simplices with each of the simplex lower neighbors.

```
INDUCTIVE(𝒱, 𝒪)
 1  V = E = ∅
 2  G = (V, E)
 3  foreach vertex v ∈ 𝒱
 4        do if 𝒪({v})
 5              ADD-NODE(G, v, 𝒪)
 6  add G as 1-skeleton to K
 7  d ← 1
 8  while d ≤ dim(K)
 9        do foreach d-simplex τ
10                 do N ← SIMPLEX-LOWER-NBRS(G, τ)
11                    foreach v_n ∈ N
12                          do σ ← τ ∪ {v_n}
13                             if 𝒪(σ)
14                                then K ← K ∪ σ
15           d ← d + 1
16  return K
```

3.2.4 Maximal Bottom-Up

MAXIMAL-BU follows the design of INCREMENTAL but is optimized to store only the maximal simplices of the complex. As a result we do not need to check and insert all facets of σ before checking σ itself. The following BUILD-MAXBU function replaces BUILD-INCREMENTAL above.

```
BUILD-MAXBU(τ, 𝒪, G, N, K)
 1  remove any facets of τ from K
 2  if N = ∅
 3      then K ← K ∪ τ
 4           return
 5  N_τ ← N
 6  foreach v_n ∈ N:
 7      do σ ← τ ∪ {v_n}
 8         if 𝒪(σ)
 9            then N' ← N_τ ∩ LOWER-NBRS(G, v_n)
10                 BUILD-MAXBU(σ, 𝒪, G, N', K)
11            else remove v_n from N_τ
```

On line 11 we remove v_n from the list of τ's neighbors because if σ does not exist then any coface of σ cannot exist due to the structural properties of the simplicial complex.

3.2.5 Maximal Top-Down

As opposed to the previous three algorithms, MAXIMAL-TD builds the complex by following a top-down approach. We again consider vertices one by one. However, for each vertex v, instead of trying to form new simplices with neighbors of v, we loop through the already existing simplices in K and try to "attach" v to each of

them. In order to do this procedure efficiently, we keep the complex in its maximal representation and go through the simplices in a decreasing order of their dimension.

MAXIMAL-TD$(V, \mathcal{O})$

```
1  K ← (K_0, K_1, ..., K_{n-1}) where each K_i = ∅
2  D ← 0
3  foreach vertex v ∈ V
4        do for d ← D down to 0
5              do foreach τ ∈ K_d
6                    do σ ← τ ∪ {v}
7                       if τ marked and σ ∈ K
8                          then K_d ← K_d − τ
9                               continue
10                      if O(σ)
11                         then K_d ← K_d − τ
12                              K_{d+1} ← K_{d+1} ∪ σ
13                              D ← max(D, d + 1)
14                         else
15                              if d ≥ 1
16                                 then mark and insert faces* f of τ in K_{|f|−1}
17                              if τ marked
18                                 then K_d ← K_d − τ
19             d ← d − 1
20       if no new simplices inserted and O({v})
21          then K_0 ← K_0 ∪ {v}
22 return K
```

Throughout, we maintain a collection of complexes K_i, each of which stores only the i-dimensional maximal simplices of K. This is particularly useful for the traver-

sal of the simplices and for maintaining the maximal representation of the complex by simplex insertion and removal. An example of the latter are the operations on Lines 11-12 in MAXIMAL-TD($\mathcal{V}, \mathcal{O}$), which get executed if $\mathcal{O}(\sigma) = T$. Similarly, if σ does not exist, we add faces of τ to lower-dimensional complexes so that in the following iterations of the `while` loop we can try to attach v to each of them. We have implemented two different protocols for choosing which faces of τ to leave for further consideration on Line 16. We refer to the two protocols as *standard* and *edge-dependent*:

- *standard* – add all facets of τ to the 1-dimension-lower complex
- *edge-dependent* – find the set f of vertices in τ with which v forms an edge and let $d' = |f| - 1$. If $d' < d$, insert the simplex f in $K_{d'}$; else, follow the standard protocol.

The choice of a protocol depends on the sparsity of the 1-skeleton of the complex and is further discussed in Section 3.3.3. Note that before inserting faces of τ in lower-dimensional complexes on Line 16, we mark them to denote that they have come from a higher dimensional simplex and are not maximal simplices. This allows us to remove them quickly in a subsequent iteration on Line 18 if they do not form a new simplex with v, thus preserving the maximal representation of the complex.

The marking of the simplices also helps in avoiding unnecessary calls to the oracle (Lines 7-9). To illustrate this, consider the following example. Let K be a complex with three maximal simplices abc, acv and abv (for ease of notation we denote a simplex $\{abc\}$ by abc). We perform the construction with the MAXIMAL-TD algorithm using the standard protocol by considering the vertices in the following order: a, b, c, v. Table 3.1 shows K_0, K_1 and K_2 after each oracle call $\mathcal{O}(\sigma)$. Note that when we try to add v, the complex has only one maximal simplex abc, and since $\mathcal{O}(abcv) = F$, we add the 3 facets of abc to K_1. Similarly, since $\mathcal{O}(bcv) = F$, b and c get inserted

#	σ	$\mathcal{O}(\sigma)$	K_0	K_1	K_2
1	a	T	a		
2	ab	T		ab	
3	abc	T			abc
4	$abcv$	F		bc ac ab	abc
5	bcv	F	b c	ac ab	abc
6	acv	T	b c	ab	abc acv
7	abv	T	b c		abc acv abv

Table 3.1: Constructing a complex K with three maximal simplices $\{abc\}$, $\{acv\}$ and $\{abv\}$ with MAXIMAL-TD using the standard protocol. Each row shows the simplices in K_0, K_1 and K_2 after calling the oracle with σ.

into K_0. However, by the time we try to attach v to any of the simplices in K_0, acv and abv have been inserted in K_2, and calling the oracle with bv or cv becomes unnecessary. Since b and c are marked, we first check if bv or cv are contained in a higher-dimensional maximal simplex and since they are, we do not make any extra calls to the oracle.

3.3 Complexity

Simplicial complexes are exponentially-sized objects and therefore, we can analyze the efficiency of algorithmic approaches for their construction only in terms of output-sensitive complexity. A thorough theoretical analysis on this matter is outside the scope of this thesis but to facilitate a discussion in this direction, we present the following results: The first one is on the performance of MAXIMAL-TD in the case when the complex achieves its maximum size, and the second one illustrates how the order in which we consider vertices in the construction affects the algorithm's

complexity. Throughout, we use the standard protocol and analyze the complexity is terms of total number of calls made to the oracle since they are considered to be the bottleneck of the computation. We then end the section by discussing the differences in the performance of Maximal-TD when using each of its two protocols.

3.3.1 Complex Size

A maximal algorithm computes the maximal simplices of the complex. Since a simplicial complex can also be viewed as a poset as defined in Section 2.1.1, the algorithm enumerates all maximal elements of the poset. Recall that maximal elements form an antichain and therefore, the maximum size of the constructed complex is the same as the size of the longest antichain in the poset. By Sperner's Theorem [11] the maximum size, or cardinality, of an antichain A on n elements is:

$$|A| \leq \binom{n}{\lfloor n/2 \rfloor} \approx \frac{2^n}{\sqrt{\pi n/2}}. \tag{3.1}$$

Consequently, a simplicial complex built on $|\mathcal{V}| = n$ vertices has $\Theta(2^n n^{-1/2})$ maximal simplices. Since each simplex has $n/2$ vertices, the size necessary to store a complex in its maximal representation becomes $\Theta((2^n n^{-1/2})n/2) = \Theta(2^n\sqrt{n})$.

From Equation 3.1 we see that a complex constructed with a maximal algorithm achieves its maximum size when it has all possible simplices of dimension $\lfloor n/2 \rfloor - 1$. Consider the number of calls made to the oracle by Maximal-TD with the standard protocol in this worst output-size case. Without loss of generality, assume that n is even. In order to build the first $(n/2-1)$-dimensional simplex, the algorithm does $n/2$ calls to the oracle: one for the first vertex, and one for each subsequent vertex getting attached to the growing simplex. Once the simplex is built, for each of the remaining $n/2$ vertices we first try to attach the vertex to all present $(n/2-1)$-dimensional simplices; then, upon failure, we form new simplices with all $(n/2-2)$-dimensional facets. Since upon arrival of the $(n/2+i)$-th vertex, $i \in [1, n/2]$, there are $\binom{n/2+i-1}{n/2}$

maximal simplices in the complex, using [13] we get that the total number of *bad* oracle calls, i.e. calls to the oracle that return F, is:

$$\sum_{j=n/2}^{n-1} \binom{j}{n/2} = \binom{n}{n/2+1}.$$

The total number of *good* oracle calls, i.e. calls to the oracle that return T, is equal to the number of maximal simplices in the constructed complex, plus the $n/2-1$ calls made before building the first maximal simplex. Therefore, in constructing the complex, MAXIMAL-TD makes a total of $\Theta(\binom{n}{n/2+1} + \binom{n}{n/2} + \frac{n}{2} - 1) = \Theta(\binom{n}{n/2})$ calls to the oracle.

3.3.2 Vertex Order

MAXIMAL-TD is an incremental algorithm and builds the complex by considering vertices one by one. We now show that its complexity may change significantly based on which vertices are considered first. Experimental evidence for this observation is further presented in Section 4.2, and its implications are demonstrated in two of the applications of our framework described in Sections 5.1 and 5.3.

Let K be a complex on n vertices $\mathcal{V} = \{v_i\}_{1 \leq i \leq n}$. Suppose that K has one d-dimensional maximal simplex σ formed on vertices v_1 through v_{d+1}, and $n - (d+1)$ 0-simplices formed by each of the remaining vertices v_{d+2} through v_n. Let $(\cdot)$ denote an ordered tuple and consider the following two cases:

1. Vertices arrive in the order (v_i) where i varies from 1 to n.

2. Vertices arrive in the order $(v_{\pi(i)})$ where i varies from 1 to n, and $\pi(i)$ is a permutation defined as:

$$\pi(i) = \begin{cases} d+1+i, & \text{if } i \in [1, n-d-1] \\ n-i+1, & \text{if } i \in [n-d, n]. \end{cases}$$

We construct K with MAXIMAL-TD using the standard protocol as follows:

Case 1 Since the first $d+1$ vertices form σ, for each of them we do a single call to the oracle as we attach it to the growing simplex. Then, when vertex v_{d+2} arrives, since it does not form a simplex with any of the faces of σ, we do $2^{d+1}-1$ *unsuccessful* calls to the oracle, i.e. calls that return F, then one more with just v_{d+2} that returns T, as the vertex forms a 0-simplex. Similarly, for each of the following vertices we do $2^{d+1}-1+c$ unsuccessful calls, trying to attach the vertex to σ and to each of the c present 0-dimensional maximal simplices, before successfully inserting it in K_0. Since $c \leq n$, we can write the total number of calls made to the oracle as $O((d+1)+(n-d-1)2^{d+1}+(n-d-1)c) = O(n^2+(n-d)2^d)$.

Case 2 Now, the first $n-d-1$ vertices form only 0-dimensional maximal simplices. As each of these vertices arrives, we try to attach it to the simplices already in K_0 before testing and inserting the 0-simplex itself, which results in a total of $\sum_{i=1}^{n-d-1} i$ calls to the oracle. For each of the remaining $d+1$ vertices, we try to form simplices with the present 0-dimensional simplices, then make one additional call to the oracle as we attach the vertex to the growing σ. Hence, the construction of the complex takes a total of $O(n^2+(n-d-1)(d+1)+(d+1)) = O(n^2)$ calls to the oracle.

The example above illustrates that the complexity of the algorithm may be $O(n^2)$ or $O(n^2+(n-d)2^d)$ depending on the order in which vertices forming higher dimensional simplices are considered in the construction. The difference in performance may be negligible or huge based on the value of d relative to n as shown in the following table:

d	Case 1	Case 2
$O(1)$	$O(n^2)$	$O(n^2)$
$O(n)$	$O(2^n)$	$O(n)$
$O(\sqrt{n})$	$O(n2^{\sqrt{n}})$	$O(n^2)$

Note that when $d = n$ the complex consists of a single maximal simplex and the algorithm performs the construction with only n calls to the oracle.

3.3.3 Protocol Choice

The efficiency of the two protocols for the Maximal-TD algorithm depends on the sparsity of the 1-skeleton of the complex: If most of the edges are present in the full complex, then the standard protocol would be more efficient than the edge-dependent one. If however, the 1-skeleton is sparse, i.e. a small fraction of all possible edges are present, then the edge-dependent protocol would be more efficient. To illustrate the difference in the performance of Maximal-TD when using one of the protocols as opposed to the other, we present the following example.

Suppose we want to test if a vertex v forms a new simplex with a d-dimensional simplex $\sigma \in K$, $v \notin \sigma$, and consider the following two scenarios:

1. v forms an edge with each vertex in σ;

2. v does not form an edge with any of the vertices in σ.

If we use the edge-dependent protocol in the fist case, we will make at most $d+1$ more calls to the oracle than if we use the standard protocol. This difference results from the additional tests of whether v forms an edge with each of the $d+1$ vertices of σ.

However, if we use the standard protocol in the second case, we will make $O(2^d)$ more calls to the oracle than if we use the edge-dependent protocol. The reason for

this is that since v does not form any edges with the vertices in σ, with the standard protocol we will essentially try to attach v to each of the $2^{d+1} - 1$ faces of σ. At the same time, with the edge-dependent protocol we will make only $d + 1$ oracle calls before concluding that v does not form any new simplices with σ and its faces.

These differences indicate that if no prior knowledge about the sparsity of the 1-skeleton exists, using the edge-dependent protocol would be a better choice when constructing the complex. Alternatively, we can sample the graph at the beginning to estimate the sparsity. Note that while the efficiency of the edge-dependent protocol depends on the density of the 1-skeleton, we can design other protocols in a similar manner in order to incorporate prior information about the sparsity of higher-dimensional skeletons of the complex as well.

Chapter 4

Experiments

In this chapter, we analyze the performance of the four algorithms described in Section 3.2. The design of the experiments follows the approach outlined in Section 3.1. Recall that $B = \{b_i\}_{1 \leq i \leq n}$ is a set of objects and $g : B \to \mathcal{V}$ is a bijective function that maps each object to a vertex in $\mathcal{V}$. Vertices in the construction are always considered in the order v_1 to v_n. Each experiment then is described in terms of:

- **object type**: type of the objects in the set B
- $\boldsymbol{n}$: total number of objects
- $\boldsymbol{b_i}$: the i-th object
- $\boldsymbol{g(b_i)}$: mapping of the objects to vertices
- $\boldsymbol{\mathcal{O}(\sigma)}$: condition upon which the oracle returns T
- **total order**: relation on the set $\mathcal{V}$ defined on the indices of the elements $v_i \in \mathcal{V}$
- **params**: parameters varied in the experiment
- **algs**: applied algorithms
- **runs/pt**: number of runs of an algorithm averaged per point

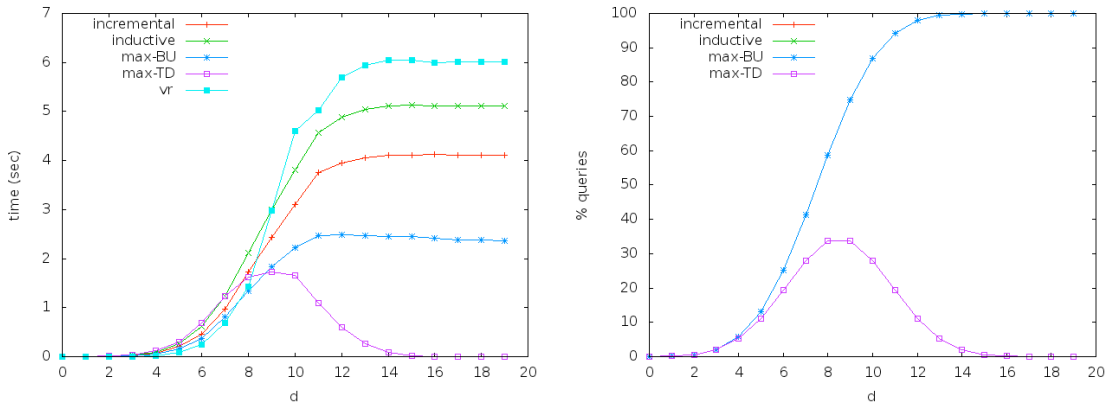
All experiments are run on a 64-bit Linux machine with 32GB of RAM and two dual-core 3GHz Xeon processors, although our software uses only one core.

4.1 Comparison

We first compare the performance of our four algorithms with each other and with the construction of the VR-complex. The input data consists of 20 vertices on which we construct complexes of different dimension. Each d-dimensional complex has all simplices of dimension up to d. Thus for a particular value of d our oracle returns T if the corresponding set of vertices has cardinality at most $d+1$, and F otherwise. For the VR-construction we use the Incremental algorithm presented in [24]. The algorithm takes as input a neighborhood graph G (in our case, with all edges

object type	number
$\boldsymbol{n}$	20
$\boldsymbol{b_i}$	$i-1$
$\boldsymbol{g(b_i)}$	v_i
$\boldsymbol{\mathcal{O}(\sigma)}$	$\lvert\sigma\rvert \leq d+1$
total order	$<$
params	$d \in [0,19]$, $d \in \mathbb{Z}$
algs	Incremental-VR from [24] Incremental Inductive Maximal-BU Maximal-TD with edge-dependent protocol
runs/pt	10

Table 4.1: Setup for Experiment 1



(a) runtime in seconds

(b) number of queries as percent of 2^{20}

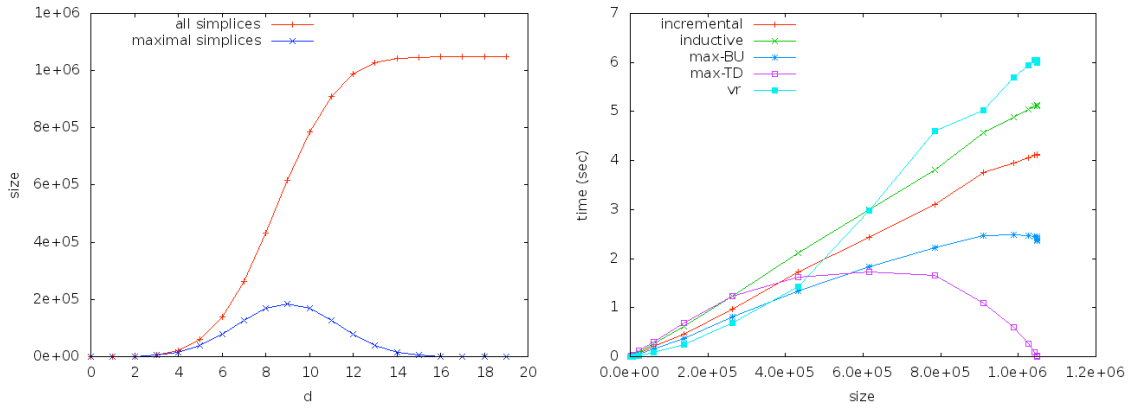
Figure 4.1: [20 vertices] Building complexes that contain all simplices of dimension up to d.

present) and builds the clique complex on G up to dimension d. Table 4.1 summarizes the setup for the experiment.

We run the five algorithms for d varying from 0 to 19 and Figure 4.1(a) shows the resulting runtime of the algorithms. As we see, their behavior changes as the dimension of the complex increases. The runtime of Maximal-TD is directly related to the number of maximal simplices in the complex illustrated in Figure 4.2(a). The runtime of the other algorithms is proportional to the size, or total number of simplices, of the complex. The particular differences in the relative performance of the algorithms is a natural consequence of their design.

Note that our algorithms outperform VR for all $d > 9$. This result is a consequence of the simplicity of our oracle: while VR checks for the existence of all pairwise edges in a given simplex, our oracle simply compares the cardinality of the set with $d+1$. Therefore, the behavior observed in Figure 4.1(a) cannot be used to make any assumptions on the general relative performance of the algorithms. It does, however, indicate that the efficiency of the oracle is directly related to the efficiency of our algorithms, as discussed in the next section.

Figure 4.1(b) shows the number of queries performed by our four algorithms.



(a) size of complex as a function of d

(b) runtime as a function of the total number of simplices in complex

Figure 4.2: [20 vertices] Size of each complex from Figure 4.1 and its relation to the algorithms' runtime.

While the behavior of MAXIMAL-TD in this case is similar to the algorithm's runtime, the other three algorithms make the same number of calls to the oracle in contrast to their differing runtimes. Moreover this number corresponds to the total number of simplices in the complex. The reason for this is the fact that each k-dimensional cell is the clique on $k+1$ vertices and thus, every vertex has k neighbors. Since the three algorithms use the neighbors of the vertices to try and form higher dimensional simplices, the number of queries that they make to the oracle is the same. Furthermore, since they are all bottom-up approaches, this number is equal to the number of simplices in the complex. Note that by the time when INCREMENTAL checks for the existence of the facets of σ on Line 4 in BUILD-INCREMENTAL, all of the facets are already in the complex and therefore, no extra calls to the oracle are made. However, the algorithm still needs to make sure that the facets are in the complex and therefore, even though INCREMENTAL and MAXIMAL-BU have the same number of queries, MAXIMAL-BU is much faster than INCREMENTAL.

Note that MAXIMAL-TD makes no more than 33.7% of the total possible number of queries. The worst case occurs when $d = 9$ and the complex has all of its $\binom{n}{n/2}$ maximal simplices. From Figure 4.2(b) we see also that as the size of the complex

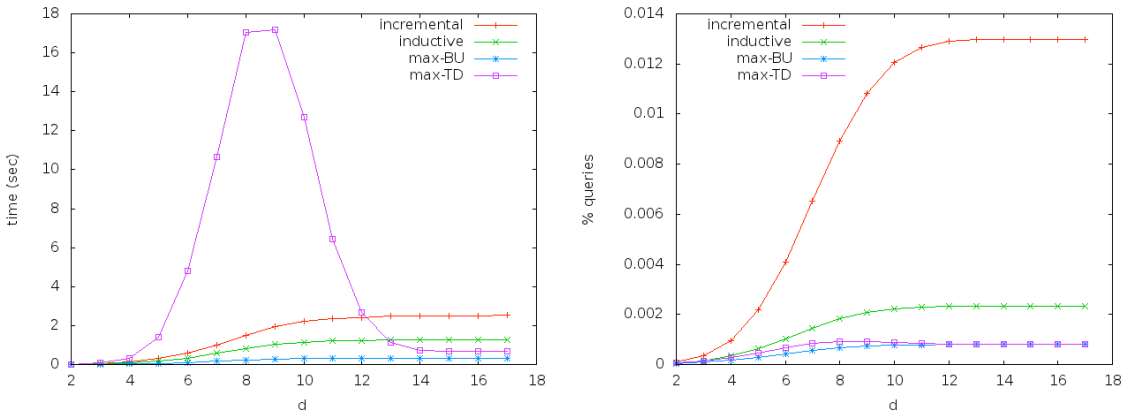
increases the runtime of both maximal algorithms decreases, MAXIMAL-TD more obviously so than MAXIMAL-BU, as opposed to INCREMENTAL and INDUCTIVE.

4.2 Arbitrary Complexes

The complexes that we build in Section 4.1 are non-geometric complexes but they contain all maximal cliques up to a certain dimension and we can construct them with other algorithms such as VR. However, our framework can be used also to construct arbitrary complexes that may not be clique complexes of their 1-skeleton. In other words, we can build complexes that may have all faces of a simplex σ without having σ itself, even if the dimension of the complex is greater than $\dim(\sigma)$. The simplicial complex in Figure 2.1(b) is an example of one such complex: even though all faces of $\{abc\}$ exist and $\dim(K') = 2$, the triangle is not present in the complex.

object type	number
$\boldsymbol{n}$	34
$\boldsymbol{b_i}$	$i - 1$
$\boldsymbol{g(b_i)}$	v_i
$\boldsymbol{\mathcal{O}(\sigma)}$	$\lvert\sigma\rvert \leq 2$, or $\lvert\sigma\rvert \leq d+1$ and $g^{-1}(v_i) \equiv 1 \pmod 2$ for all $v_i \in \sigma$
total order	$<$
params	$d \in [2, 17]$, $d \in \mathbb{Z}$
algs	INCREMENTAL INDUCTIVE MAXIMAL-BU MAXIMAL-TD with edge-dependent protocol
runs/pt	10

Table 4.2: Setup for Experiment 2



(a) runtime in seconds

(b) number of queries as percent of 2^{34}

Figure 4.3: [34 vertices] Building complexes that contain all possible 1-dimensional simplices and all simplices of dimension up to d formed on the odd-numbered vertices.

To compare the performance of our algorithms when building such arbitrary complexes we conduct the experiment summarized in Table 4.2. The input data consists of 34 vertices corresponding to the numbers 0 to 33. Each d-dimensional complex has all $\binom{34}{2}$ 1-dimensional simplices and all simplices of dimension up to d formed only on the odd-numbered vertices. Thus, for a given d, our oracle returns T if for each vertex v in the corresponding set $g^{-1}(v)$ is an odd number, and the cardinality of the set is at most $d+1$.

We run the four algorithms for d varying from 2 to 17 and Figure 4.3 shows the resulting graphs. Note that the total number of calls to the oracle for all algorithms is less than 0.02% of 2^{34}. Furthermore, Maximal-BU performs at most 6.1% of the queries that Incremental invokes. This behavior is due to the fact that not all facets of every σ exist in the complex any more so Incremental ends up making extra calls to the oracle, calls that Maximal-BU does not make since it stores only the maximal simplices.

The runtime of the four algorithms exhibits a similar relationship to the size of the simplicial complex, as in Section 4.1. However, now Maximal-TD seems to have a much poorer performance in comparison to the other three algorithms, in contrast

to previous observations. At the same time, INCREMENTAL which does up to 93.8% more calls to the oracle, runs 92.3% faster when $d = 8$. The reasons for these big differences are inherent to the design of MAXIMAL-TD and the speed of the oracle. Since all 1-dimensional simplices are present in the complex but only a subset of them form higher dimensional cliques, with MAXIMAL-TD we end up checking most of the faces of each simplex for which the oracle returns F. However, many of them already exist in the complex and these extra checks do not result in additional calls to the oracle but affect the runtime of the algorithm. (Note that the behavior of the algorithm changes when vertices arrive in a different order, as demonstrated in Section 4.3.) At the same time, INCREMENTAL checks and queries each simplex in the complex and given the simplicity of the oracle, this results in a relatively small runtime and large number of queries. However, in practice calls to the oracle are often more expensive. For example, in the application in Section 5.1 each oracle call is a query to Google search that may take up to 1 second. Consequently, the relative performance of the algorithms changes significantly. To illustrate this experimentally, we assume that the oracle is slow and model our assumption by introducing a delay of 1 millisecond to each oracle call. Figure 4.4 shows the respective runtimes.

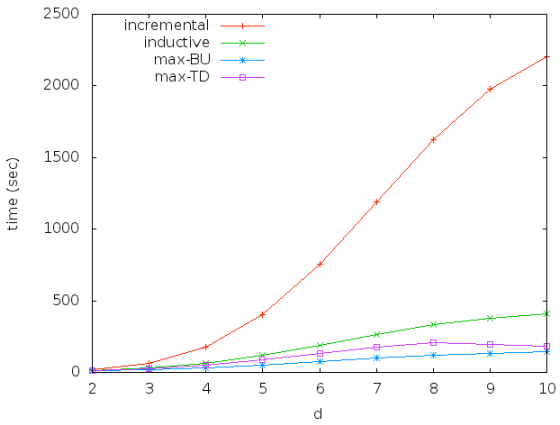


Figure 4.4: [34 points] Same experiment as in Figure 4.3 with oracle delay of 10^{-3}s.

Note that we run MAXIMAL-TD with the edge-dependent protocol even though all edges are present in the 1-skeleton of the complex. We do this in order to keep our analysis general and avoid bias in the results. However, we should point out that

in this particular experiment the difference in performance between the two protocols is negligible: the runtime of the algorithm with the standard protocol differs with no more than 1.07 seconds from the presented runtime (with the largest difference occurring when the complex achieves its maximum size at $d = 8$).

4.3 Vertex Order

The performance of MAXIMAL-TD is affected by the order in which vertices forming higher-dimensional simplices are considered in the construction, as observed in Section 3.3.2. To experimentally demonstrate this dependence we use the setup created in Section 4.2 but with two additional oracles. Recall that in the original experiment

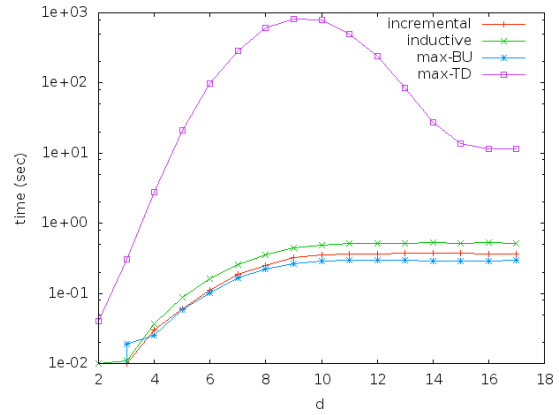
object type	number
$\boldsymbol{n}$	34
$\boldsymbol{b_i}$	$i - 1$
$\boldsymbol{g(b_i)}$	v_i
$\boldsymbol{\mathcal{O}(\sigma)}$	beg: $\lvert\sigma\rvert \leq 2$, or $\lvert\sigma\rvert \leq d+1$ and $g^{-1}(v_i) \leq 16$ for all $v_i \in \sigma$
	mix: $\lvert\sigma\rvert \leq 2$, or $\lvert\sigma\rvert \leq d+1$ and $g^{-1}(v_i) \equiv 1 \pmod 2$ for all $v_i \in \sigma$
	end: $\lvert\sigma\rvert \leq 2$, or $\lvert\sigma\rvert \leq d+1$ and $g^{-1}(v_i) > 16$ for all $v_i \in \sigma$
total order	$<$
params	$d \in [2, 17]$, $d \in \mathbb{Z}$
algs	INCREMENTAL INDUCTIVE MAXIMAL-BU MAXIMAL-TD with edge-dependent protocol
runs/pt	10

Table 4.3: Setup for Experiment 3

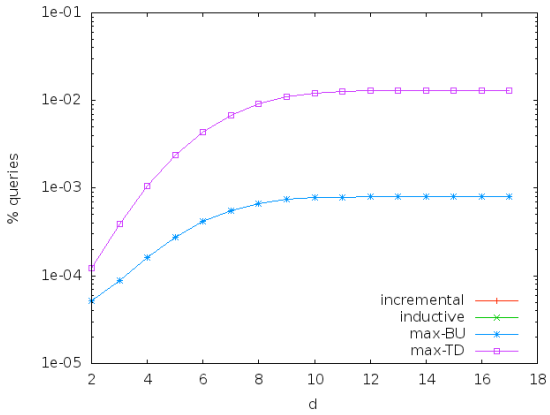
we build higher dimensional simplices only on vertices corresponding to odd numbers. There are exactly 17 such vertices and since $g(b_i) = v_i$, they are "mixed" with the vertices corresponding to even numbers and do not arrive, i.e. are considered in the construction, in a consecutive order. We refer to this original oracle as $\mathcal{O}_{\text{mix}}$. In addition, we introduce two other oracles, $\mathcal{O}_{\text{beg}}$ and $\mathcal{O}_{\text{end}}$, that allow us to instead build higher-dimensional simplices on the first and last 17 vertices respectively, as shown in Table 4.3. Note that the complexes constructed with the three oracles are identical. The only difference is in the order in which we consider vertices forming higher-dimensional simplices.

Figure 4.5 shows the performance of the algorithms with each of the three oracles, in terms of both runtime and number of queries. First, consider the behavior of Maximal-TD. The algorithm is slowest when the vertices forming higher dimensional simplices arrive at the beginning, and fastest when they come at the end, which confirms our theoretical observations from Section 3.3.2. Consequently, with $\mathcal{O}_{\text{beg}}$ Maximal-TD is up to 4 orders of magnitude slower than the other three algorithms while with $\mathcal{O}_{\text{end}}$ it is up to 3 orders of magnitude faster than Incremental, for instance. Maximal-BU, on the other hand, is insensitive to the order of the vertices and always outperforms the full representation algorithms. It is faster than Maximal-TD as well, in all cases except when $d > 7$ with $\mathcal{O}_{\text{end}}$. This behavior indicates that the most efficient way to construct such arbitrary complexes is by using Maximal-BU for lower dimensional complexes and Maximal-TD for higher dimensional ones, provided we have an optimal ordering of the vertices. If no such ordering is available, then Maximal-BU seems to be the best choice.

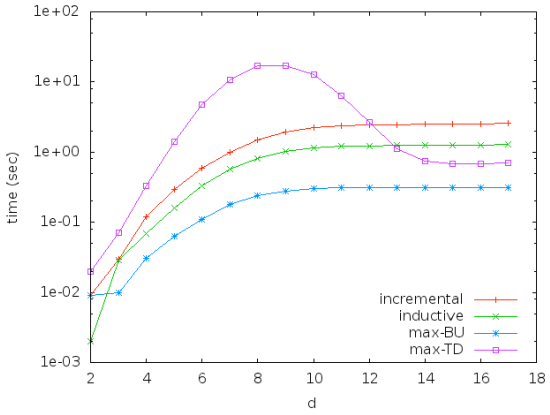
Another interesting observation from the graphs is that the performance of Incremental also seems to depend on the order of the vertices. However, as opposed to Maximal-TD, the algorithm performs best when vertices forming higher dimensional simplices arrive at the beginning, and worst when they arrive at the end.



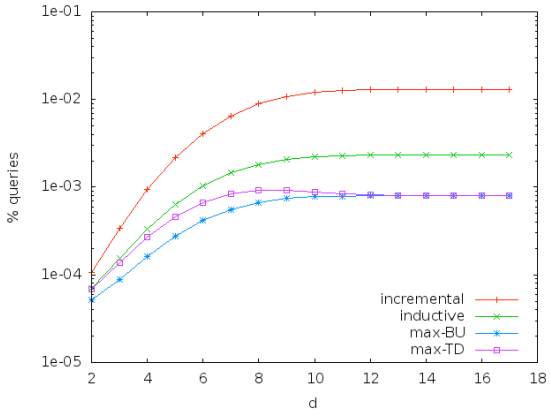
(a) runtime, $\mathcal{O}_{\text{beg}}$



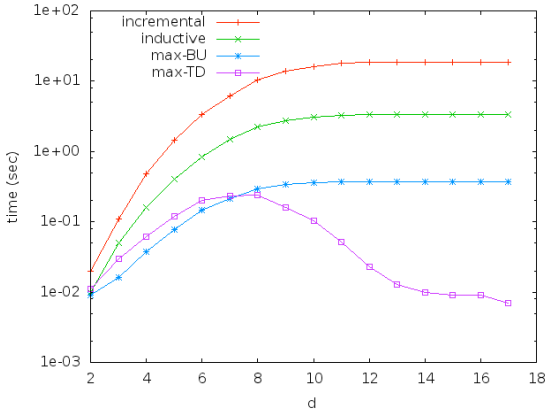
(b) queries as percent of 2^{34}, $\mathcal{O}_{\text{beg}}$



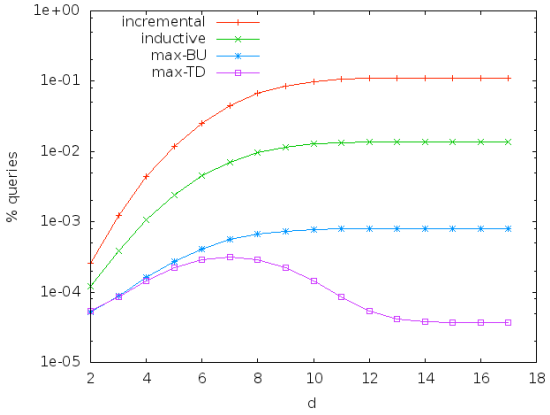
(c) runtime, $\mathcal{O}_{\text{mix}}$



(d) queries as percent of 2^{34}, $\mathcal{O}_{\text{mix}}$

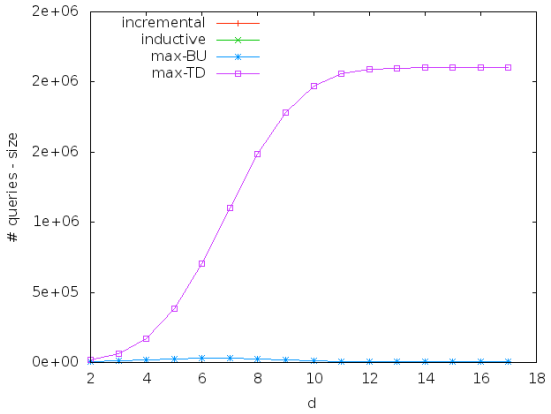


(e) runtime, $\mathcal{O}_{\text{end}}$

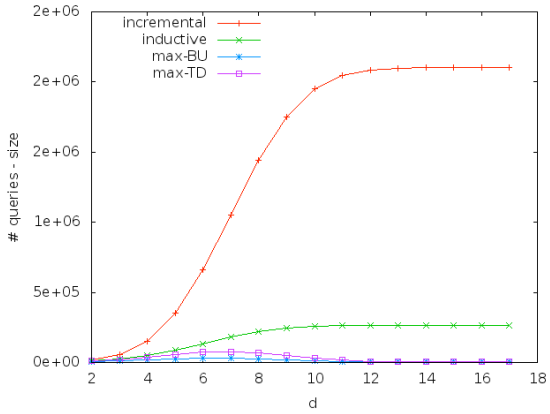


(f) queries as percent of 2^{34}, $\mathcal{O}_{\text{end}}$

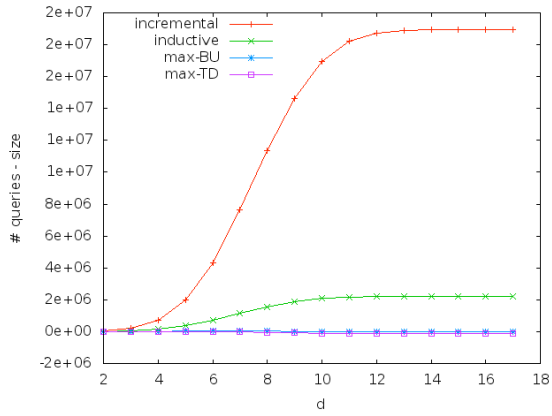
Figure 4.5: [34 vertices] Building complexes with three different oracles: $\mathcal{O}_{\text{beg}}$, $\mathcal{O}_{\text{mix}}$ and $\mathcal{O}_{\text{end}}$ as defined in Table 4.3: log graphs of runtime and actual number of queries as percent of possible number of queries, when vertices forming higher-dimensional simplices are considered at the beginning ((a) and (b)), in a mixed order all throughout ((c) and (d)), or at the end of the construction ((e) and (f)) respectively.



(a) difference with $\mathcal{O}_{\text{beg}}$



(b) difference with $\mathcal{O}_{\text{mix}}$



(c) difference with $\mathcal{O}_{\text{end}}$

Figure 4.6: [34 vertices] Building complexes with three different oracles: $\mathcal{O}_{\text{beg}}$, $\mathcal{O}_{\text{mix}}$ and $\mathcal{O}_{\text{end}}$ as defined in Table 4.3: displaying the difference between the number of oracle calls and the size of the full complex when vertices forming higher-dimensional simplices are considered at the beginning, in a mixed order all throughout, or at the end of the construction respectively.

To gain an intuition for the reason for this behavior, consider a d-dimensional simplex σ. While building σ, for each of its faces $\tau \subseteq \sigma$, the algorithm first tests if τ exists, then tries to form new simplices with τ and each of its lower neighbors. If σ is formed towards the beginning of the construction when it has fewer lower neighbors, the algorithm thus makes fewer calls to the oracle than if σ is formed towards the end of the construction when it has more lower neighbors. Therefore, the runtime of Incremental increases as vertices forming higher dimensional simplices are

considered later in the construction.

Throughout, the number of queries stays very low and seems to correlate with the runtime of the algorithms. The only exception is with INCREMENTAL and MAXIMAL-TD when using $\mathcal{O}_{\text{mix}}$ as discussed in Section 4.2. Since the algorithms are output sensitive, the number of simplices in the constructed complex gives a lower bound on the number of calls made to the oracle. An algorithm then, is faster or slower depending on how many extra calls to the oracle it makes. Note that after a simplex is formed and added to the complex, we never again ask the oracle if that simplex exists. However, since we do not keep track of the simplices that do not exist in the complex, we may ask the oracle multiple times for each of them. Therefore, the relative performance of the algorithms is determined by these extra number of calls made to the oracle. Figure 4.6 shows this number as a function of the dimension of the complex. Note that even though the two maximal algorithms store only the maximal simplices of the complex, with $\mathcal{O}$beg and $\mathcal{O}$mix they still end up querying most of the simplices in the full complex. Only with $\mathcal{O}$end we see a significantly improved lower number of queries. Note also that in its worst case in Figure 4.6(a), MAXIMAL-TD does an order of magnitude better than INCREMENTAL does in its worst case in Figures 4.6(c).

4.4 Maximal Representation

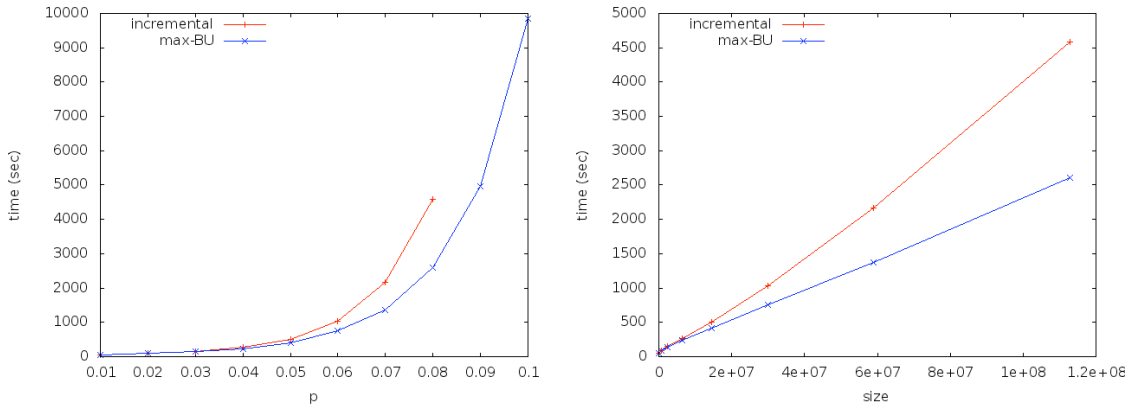
In all our experiments MAXIMAL-BU consistently outperforms INCREMENTAL. Recall that the two algorithms follow the same design but MAXIMAL-BU stores only the maximal simplices in the complex, while INCREMENTAL keeps the complex in its full representation. To further investigate the effect of the different representations on the performance of the algorithms, we generate large sparse graphs, and construct clique complexes on them. The input data consists of neighborhood graphs generated on 10,000 vertices using the $G(n, p)$ *Erdős-Rényi random graph model* [16]. In this

object type	number
$\boldsymbol{n}$	10,000
$\boldsymbol{b_i}$	$i-1$
$\boldsymbol{g(b_i)}$	v_i
$\boldsymbol{\mathcal{O}(\sigma)}$	$(v_i, v_j) \in G.E$ for all $v_i, v_j \in \sigma$, $i \neq j$
total order	$<$
params	$G = G(n, p)$: Erdős-Rényi random graph model [16] $p \in [0.01, 0.1]$, $\Delta p = 0.01$
algs	INCREMENTAL MAXIMAL-BU
runs/pt	2

Table 4.4: Setup for Experiment 4

model G is built on n vertices and an edge between any pair of vertices exists with probability p independently of the other edges. The Erdős-Rényi random graphs represent arbitrary connectivity and provide an excellent model of graphs without any geometry. Given one such neighborhood graph, our oracle returns T for any subset of the vertices that form a clique in the graph. Table 4.4 summarizes the setup for the experiment.

We run INCREMENTAL and MAXIMAL-BU for p varying from 0 to 0.1. Figure 4.7 shows the corresponding graphs. The runtime of the two algorithms depicted on Figure 4.7(a) grows exponentially, with MAXIMAL-BU growing with a slower rate than INCREMENTAL. For $p = 0.08$ the neighborhood graph has 4,001,664 edges and in building the complex, INCREMENTAL takes about 32GB of memory, thus computation for larger p values becomes infeasible. However, with MAXIMAL-BU we are able to construct complexes with p up to 0.1 when the neighborhood graph has close to 5 million edges.



(a) runtime as a function of p (b) runtime as a function of complex size

Figure 4.7: [10000 vertices] Full versus maximal representation. Building clique complexes on neighborhood graphs generated by the $G(n, p)$ Erdős-Rényi random graph model.

p:	0	0.1	0.02	0.03	0.04	0.05	0.06	0.07
I	10^4	6.8×10^5	2.4×10^6	6.3×10^6	1.4×10^7	3×10^7	5.9×10^7	1.1×10^8
M	10^4	3.5×10^5	1.2×10^6	3.7×10^6	7.3×10^6	1.2×10^7	2.1×10^7	4.3×10^7
%	100	51.61	53.80	59.21	50.66	40.67	36.8	38.14

Table 4.5: Size of complex computed by each of two algorithms, Incremental (I) and Maximal-BU (M), and percent of maximal simplices in full representation.

Figure 4.7(b) shows the runtime of the algorithms as a function of the size of the constructed complex (in its full representation). The fact that both dependencies are linear and Maximal-BU has a smaller slope than Incremental indicates that building the complex in its maximal form is more efficient than building it in its full form. We see this also from Table 4.5 which lists the number of simplices, or cells, in the complexes computed by the two algorithms and the percentage of the maximal simplices from the total number of simplices. By storing only the maximal simplices of the complex we reduce the space required for the computation by up to 61.86%.

Chapter 5

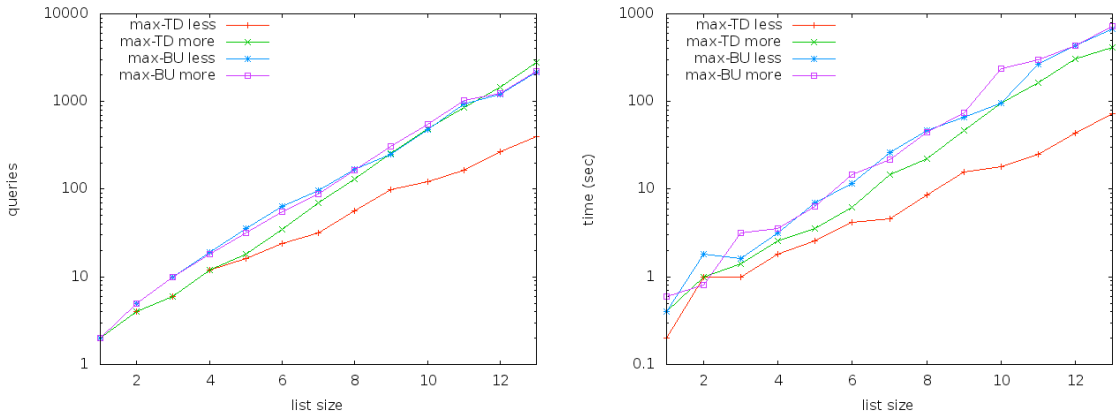
Applications

In this chapter, we present three different applications of our oracle-based framework: The first one goes back to our motivating problem of querying the web; the second utilizes the framework to create a computational model of a language, and in the third we build simplicial complexes to examine structural similarities between proteins.

5.1 Google Search

We begin with our motivating problem: finding a subset of query terms that results in a small but nonzero set of hits in Google search. To demonstrate the application of our framework we create a list of 20 grocery items and restrict our searches to the `http://recipesindian.com` domain. The table on the left in Figure 5.2 shows the items and their corresponding number of hits. We then run tests on subsets of the items with cardinality ranging from 1 to 20. Each subset of n elements is deterministically defined by the first n items from the list. Table 1 in the Appendix summarizes the setup for the application.

Our Google oracle takes a number of items, queries Google with all of them and returns T or F depending on whether the query results in any hits. Since a webpage containing k of the query terms also contains any subset of them, the oracle



(a) number of queries (b) runtime

Figure 5.1: Log graphs, comparing Maximal-BU and Maximal-TD with standard protocol when considering items in an increasing (less) and decreasing (more) order of their number of hits when queried on Google search.

is guaranteed to be monotonic.

Due to the high computational cost of the queries, we want to select an algorithm that performs a minimum number of calls to the oracle. Since Maximal-TD and Maximal-BU have consistently outperformed Incremental and Inductive in that respect, we run our tests with the first two and compare their performance. Taking into account the design of the algorithms and the nature of the data, at the beginning of each algorithm we sort the query terms by their number of hits in order to achieve better performance. Throughout this section `less` and `more` refer to the increasing and decreasing sorting respectively. Maximal-TD is implemented with the standard protocol in order to account for the high probability that a query with any two items results in a nonzero number of hits.

Figure 5.1 shows the runtime and number of queries for the two algorithms. Each data point represents the average of 3 to 5 runs of the algorithms. The unequal number of runs averaged per point results from the presence of outliers as discussed below, and the existence of a time gap between the collection of the data and its analysis. Across all of the tests the largest total runtime is in the order of hundreds of seconds

while the actual processor time is less than a second. This fact demonstrates the high dependability of the algorithms' performance on the speed of the oracle and thus, the number of queries made. Since MAXIMAL-TD with increasing initial sorting of the items achieves best runtime statistics, we use it to further analyze our data.

The table on the right in Figure 5.2 shows a summary of the results. If l is the list size, the number of *suggestions* in the 2nd column is the number of maximal simplices in the complex corresponding to all maximal combinations of the l items. In other words, these are all of the computed combinations of the items that result in a small but nonzero number of hits. The largest one from these combinations contains *q_size** grocery items and the maximum number of hits achieved across all computed combinations is displayed in the 4th column as *q_hits**. As we see, except for the trivial case when $l = 1$, each computed combination results in no more than 12 hits. The corresponding computation time is given in the last column. Note that the actual number of queries made is significantly lower than 2^l as shown in Figure 1.1.

These results show the applicability and efficiency of our framework. All computed combinations result in a small nonzero number of hits. The analysis of tests pertaining to web search queries however, should take into account several factors. First, data often contains large variability. Since Google is both updating its web index as well as changing its search algorithm, the same query may result in a different number of hits at different times. Most of the data presented in this thesis, for example, was collected in January 2012 and the query "site:recipesindian.com cauliflower beans chicken cheese tomato milk" which resulted in 1 hit then, returns 0 hits as of April 2012. Note that the number of hits in the two tables in Figure 5.2 were also collected in April 2012 and may be slightly different than the number of hits associated with the data obtained in January.

Furthermore, even though we use a license from University Research Program for Google Search to do automated queries to the search engine, the time for a single

query term	# hits
green pepper	24
cheese	174
milk	352
beans	85
tomato	245
carrot	88
cauliflower	78
chicken	178
egg	171
onion	414
potato	177
broccoli	14
garlic	389
pasta	39
zucchini	20
mushroom	34
rice	271
red pepper	23
chillies	428
pork	27

list size	# suggestions	q_size*	q_hits*	time
1	1	1	24	0.2
2	1	2	4	1
3	1	3	3	1
4	3	3	11	1.8
5	3	4	9	2.6
6	4	5	12	4.2
7	5	6	12	4.6
8	10	6	7	8.6
9	19	6	7	15.8
10	19	7	6	18.2
11	19	9	4	24.8
12	30	9	4	43.2
13	37	9	4	72
14	45	10	6	100.8
15	56	10	6	147.6
16	65	11	6	247.4
17	84	11	7	500.2
18	97	12	7	745.2
19	136	12	7	2523.4
20	142	12	7	3804.2

Figure 5.2: **Left:** list of grocery items and their corresponding number of hits; **Right**: results from running MAXIMAL-TD with standard protocol on subsets of the list: number of computed combinations (2nd column), size of the largest combination (3rd column), max number of hits across all computed combinations (4th column) and average time in seconds for each test case (5th column).

query varies a lot due to different factors. In one of our earlier tests, for instance, the total time spent in queries over three different runs with the same input parameters and the same number of queries made was 151s, 624s and 1215s. This fact necessitates the performance of a larger number of runs and removal of outliers in order to obtain a more accurate average of a given statistic with such high variability.

Aside from data variability, we need to also be aware of the quality of the results returned by Google. One of the main reasons that we restrict our search to a particular domain is because in a general Google search even a query with all 20 items from our list results in almost a million hits. Furthermore, the approximate number of results computed by Google is in most cases overly estimated and inaccurate. Since in our

current implementation of the Google oracle we check only whether the number of hits is different than 0, this issue does not have a significant effect on our computation. However, if we want to store the actual number of hits for each query and use it as a stopping criteria for building the simplicial complex, we need to take the accuracy of the Google results into account.

5.2 Languages

As a second application we apply our oracle-based framework to the construction of computational models of languages. Let S be the set of all words in a given language. Each letter from the language alphabet forms a cover set on S defined by all words which contain that letter. The nerve of this cover then captures the interaction of the letters in the space of the language dictionary.

In this experiment we use our framework to construct the nerves for English, French and Bulgarian. Our dictionary in each case is the Linux word list for the corresponding language. Following is a summary of the data. Note that the alphabets of English and French are composed of the same set of letters.

language	alphabet size	Linux word list
English	26	479,829
French	26	139,719
Bulgarian	30	823,209

Our Language oracle takes a set of letters and returns T if there exists at least one word in the language dictionary that has all of these letters. Since a word that contains a set of letters also contains any subset of the letters, the oracle is guaranteed to be monotonic. Table 2 in the Appendix summarizes the setup for the application.

We construct the corresponding nerves with the Maximal-BU algorithm. Table 5.1 shows a summary of the computed statistics for the three complexes. Even though the nerves for English and French are of the same dimension, the former is much larger in size than the latter. English, for instance has 6 cells of dimension 15

Table 5.1: Statistics for the complexes built on English, French and Bulgarian.

	dimension	#maximal cells	total # cells	time(sec)
English:	15	6,641	2,088,610	14,844
French:	15	1,845	523,975	2,231
Bulgarian:	16	11,587	4,776,802	198,585

while French has only 1. This difference in the two models may be caused by the incompleteness of the French dictionary, as hinted by the smaller size of its Linux word list, or it may be inherent to the structure of the languages. An interesting observation is that English has two 4-dimensional maximal simplices formed by {b,e,q,t,w} and {f,u,w,y,z}. This implies that if we add any other letter to either of these two subsets we cannot form a valid word from the dictionary. At the same time French has a single 3-dimensional maximal simplex while Bulgarian, like English, has two 4-dimensional ones. Note also that the complex for Bulgarian is the largest of the three but the Bulgarian Linux word list has size almost twice that of the English dictionary and its alphabet consists of 30 characters.

The results show that we can indeed apply our framework to build computational models of languages. The information encoded in the constructed nerves reveals structural dependencies among the words from the corresponding dictionaries. In addition, the topology of the models can further be analyzed to study other language characteristics or relationships. After building the two complexes in our experiment, for instance, we discovered that they both share a common subcomplex of dimension 12 which may present an interesting avenue for future topological analysis. Note however, that English and French have the same alphabet and the comparison between their models is straight-forward. Bulgarian on the other hand, has a Cyrillic alphabet of different size and we cannot compare its simplicial complex directly to the models of English and French. Such analysis would require a map from one alphabet to the other.

Since the alphabet of a language determines the cover on the words in the dic-

tionary, thus the structure of the constructed nerve, we can build different models of the same language if the set of alphabetic characters changes. This may be possible in languages such as French that contain accents or other diacritics. The precise construction then depends on whether these diacritic letters are considered separate characters or not. The French word "abîmés" which means "damaged", for example, may contain i and e according to one model and not, according to another. In our experiment we do not distinguish between accented and not accented letters.

5.3 Protein Structure

We now apply our framework to examine the structural space of proteins. Proteins are macromolecules that perform a variety of important biological functions. They are composed of chains of amino acids that fold into compact 3-dimensional shapes. Since protein structure determines protein function, the ability to find structural similarities between proteins is important for understanding their structure/function relationship. However, while there are only twenty amino acids commonly occurring in nature, the space of all possible 3-dimensional protein structures is rather complex and studying it extensively presents a challenge. Recent work at the Grigoryan lab addresses this problem by identifying a set of local structural *motifs*, or fragments, highly recurrent in proteins drawn from the Protein Data Bank (PDB) [1]. Containing over 70,000 entries, the PDB is the main repository of experimental protein structures [3]. Each identified fragment is a 3-dimensional piece of protein structure

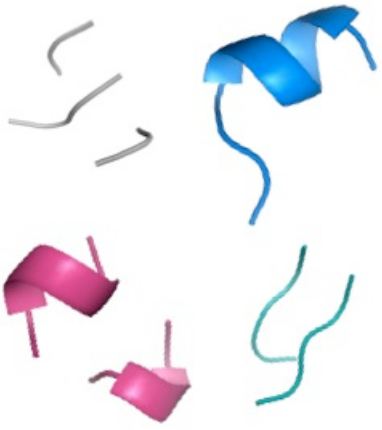


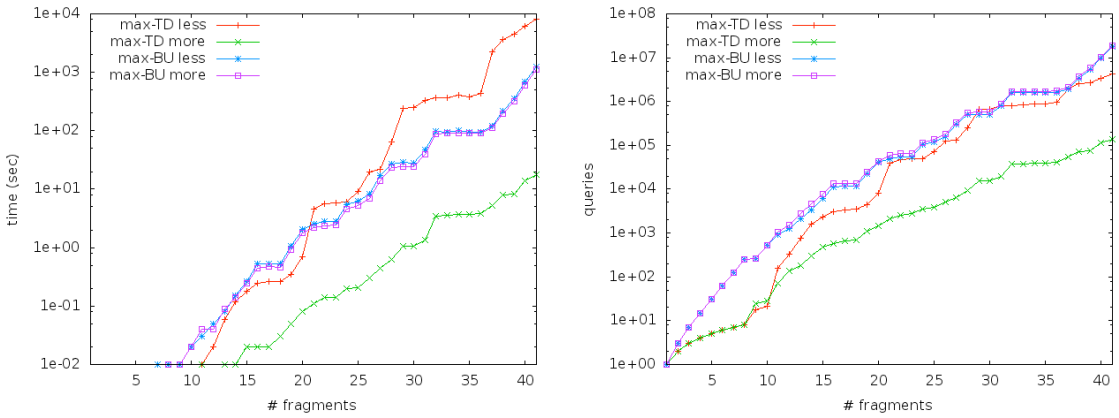
Figure 5.3: Protein fragments

that covers a number of tertiary contacts between amino acids in the protein (Figure 5.3). These fragments can thus be seen as building blocks of the molecule, or letters of its structural alphabet.

Let S be a set of proteins. Each motif forms a cover set on S defined by the proteins that contain it. The nerve of this cover then captures the relationship between the fragments in the protein structure space. Consequently, the combinatorial models encode all possible combinations of the fragments occurring together in protein structures, and can be used to extract different types of information as demonstrated below.

Our input data consists of 60 motifs $\mathcal{M} = \{m_i\}_{1 \leq i \leq 60}$ covering 20% of the structure space of 100 proteins from the PDB. Each motif $m_i \in \mathcal{M}$ has a corresponding list of proteins P_i that contain m_i. For a given subset of the motifs $M_J = \{m_j \mid j \in J\}$ where $J \subseteq \{i\}_{1 \leq i \leq 60}$, our oracle, $\mathcal{O}_N$, returns T if the intersection of the corresponding motifs' protein lists is of size at least N, i.e. $|\bigcap_{j \in J} P_j| \geq N$. A complex K_N^M then, is constructed on the set M of structural motifs with the oracle $\mathcal{O}_N$. Since if at least N proteins share a set of motifs, all of them also share any smaller combination of the motifs, the monotonicity of the oracle is guaranteed. Note however, that the oracle does not capture multiple motif occurrences within a structure. Table 3 in the Appendix summarizes the setup for the application.

Let c_i be the number of tertiary contacts covered by m_i across all considered proteins. In order to choose the most efficient algorithm, we build complexes on sets of fragments with $N = 1$ and cardinality varying from 1 to 41, by considering fragments in the construction in an increasing (`less`) or decreasing (`more`) order of their corresponding c values. We build the complexes with both, Maximal-BU and Maximal-TD. The latter is used with the edge dependent protocol since we do not have prior knowledge about the probability that any two fragments occur together in any given protein structure. Figure 5.4 shows the performance of the two algorithms



(a) runtime (b) number of queries

Figure 5.4: Log graphs, comparing MAXIMAL-BU and MAXIMAL-TD with edge-dependent protocol when fragments are considered in an increasing (less) and decreasing (more) order of number of covered tertiary contacts.

with each of the two orderings. The differences in the behavior of MAXIMAL-TD indicate that the number of contacts covered by the fragments is related to the tendency of the fragments to form higher dimensional simplices. In particular, it shows that motifs covering a larger number of tertiary contacts are more likely to appear together with other motifs in a given protein structure. Since MAXIMAL-TD `more` has the best runtime, we construct all complexes in our subsequent analysis with it by considering the fragments in a decreasing order of their c values.

We first construct $K_1^{\mathcal{M}}$. Recall that $\mathcal{M}$ is the set of all 60 motifs. The construction takes 47 minutes and the complex has 67 maximal simplices, the largest one of which is of dimension 27. Note that even though there are more than 2^{28} simplices, we are able to build the complex by storing only the maximal ones. Table 5.2 shows the number of cells, or simplices, in each dimension. Each maximal simplex represents a "unique" combination of the motifs that occurs in a single protein and is not shared with any other protein. The fact that there is one maximal simplex of dimension 27 but none of dimension 25 and 26, for instance, indicates that all combinations of 25 and 26 fragments that can be found together in protein structures, are subsets of the 28

dim	0	1	2	3	4	5	6	7	8	9	10	11	12	13
cells	1	1	2	0	0	1	1	2	10	6	5	1	5	4
dim	14	15	16	17	18	19	20	21	22	23	24	25	26	27
cells	7	5	2	3	4	1	1	1	0	2	1	0	0	1

Table 5.2: Complex $K_1^{\mathcal{M}}$

dim	0	1	2	3	4	5	6	7	8	9
cells	1	1	0	4	5	27	44	38	36	30
dim	10	11	12	13	14	15	16	17	18	19
cells	26	26	22	14	8	3	2	1	2	1

Table 5.3: Complex $K_2^{\mathcal{M}}$

fragments from the 27-dimensional simplex. Furthermore, the proteins corresponding to the maximal simplices are all different from each other, which shows that 67 out of the 100 proteins considered in the study contain a unique maximal subset of the motifs.

We now construct $K_2^{\mathcal{M}}$. The complex has 280 maximal cells, each of which corresponds to a set of motifs shared by at least two proteins. Table 5.3 shows the number of cells in each dimension. The complex has dimension 19 and its construction takes less than 17 seconds. All maximal simplices except one of the 7-dimensional ones, correspond to combinations of motifs shared by exactly 2 proteins. To visualize these relationships, we build a graph G in which each node is a protein and two nodes are connected with an edge if the two proteins share at least one fragment. The thickness of the edges indicates the number of the shared fragments. Figure 5.5 shows a graph of all proteins sharing 15 or more fragments. As we see some proteins like 1GSOA and 1EKQA share fragments with many other proteins, while others like 1BYIA and 1CXQA share fragments only with 1GSOA. Not surprisingly, 1GSOA and 1EKQA are the two proteins sharing the largest combination of motifs in $K_2^{\mathcal{M}}$ (the 19-dimensional simplex), and 1GSOA itself is the protein that contains the largest unique combination of motifs (the 27-dimensional simplex in $K_1^{\mathcal{M}}$).

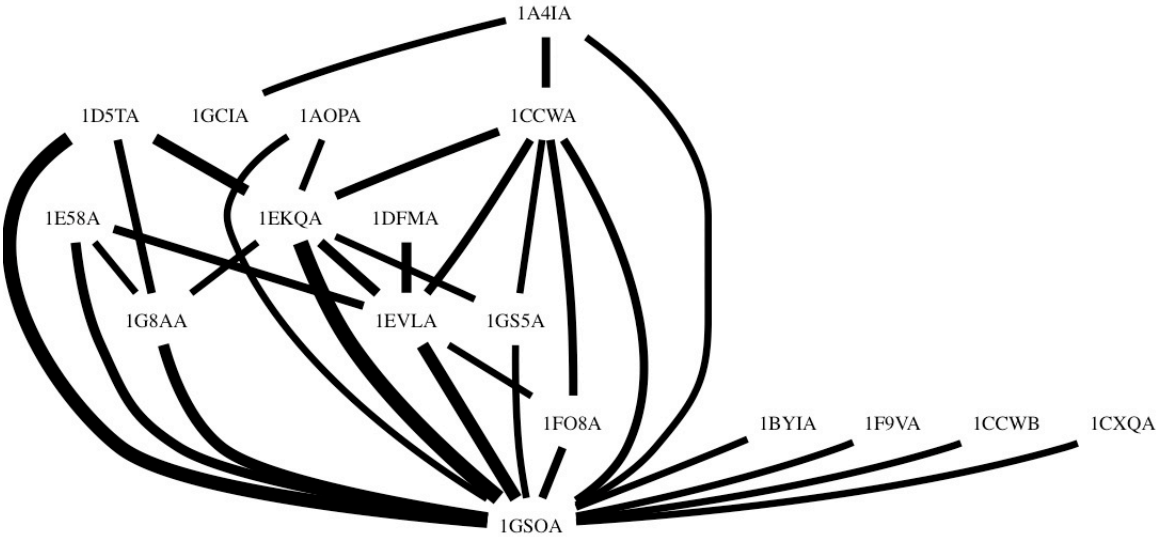


Figure 5.5: Graph of pairwise protein relationships built from $K_2^{\mathfrak{M}}$. Thickness of the edges indicates number of shared fragments. Displayed are only proteins sharing 15 or more fragments.

Building complexes on all 60 motifs gives us a larger picture of how the 100 proteins are related. However, we often want to find *structural neighbors* of a particular protein, that is, proteins that exhibit structural similarity to a given query protein. Different criteria can be used to define the degree of similarity between protein structures. In the context of our work this criteria is the number of shared motifs between the proteins. In order to find and rank the structural neighbors of a query protein then, we do the following: Let P be the query protein. We first find the set M^P of all motifs that occur in P, then build the complex $K_2^{M^P}$ on them. Note that $K_1^{M^P}$ always consists of a single simplex – the one corresponding to M^P – and does not contain any interesting information. $K_2^{M^P}$ however, has as its maximal simplices the largest combinations of fragments shared between P and at least one other protein. Thus, having built the complex, we can rank the proteins by the number of fragments they share with P, as indicated by the dimension of the corresponding maximal cells in the complex.

Following this procedure, we use 1D5TA as our query protein and compute a

rank	# shared fragments	proteins sharing fragments with 1D5TA
1	19	1GSOA
2	17	1EKQA
3	15	1G8AA
4	13	1CXQA
5	12	1F9VA, 1DJ0A, 1GS5A
6	11	1GK9B
7	10	1FX2A, 1GPQA
8	9	1D4OA, 1CC8A
9	8	1F8EA, 1F86A
10	7	1A62A
11	6	1GKMA, 1GP0A, 1AH7A, 1EB6A
12	5	1G3PA
13	2	1FD3A

Table 5.4: Ranking proteins according to their structural similarity to 1D5TA. Number of shared fragments corresponds to the dimension of the maximal simplices in $K_2^{M^P}$ where M^P is the set of fragments contained in 1D5TA.

ranking for the proteins based on how many fragments they share with it. The complex has 20 maximal simplices and Table 5.4 shows the resulting ranking. First, note that 1D5TA contains a total of 24 fragments from $\mathcal{M}$ but the largest combination of fragments that it shares with another protein (1GSOA) is 19. Since different proteins may share the same number of fragments with the query protein, multiple structures may be given the same rank, as in the case with 1F8EA and 1F86A, for instance. In general, we expect that a protein sharing a larger number of motifs with the query protein would be more similar to it than a protein sharing a smaller number of motifs. As a way of testing this hypothesis, on Figure 5.6 we display the structures of 1EKQA and 1FD3A with which 1D5TA shares 17 and 2 fragments respectively. We see that 1EKQA does indeed look more similar to 1D5TA than 1FD3A. To further investigate the structural similarities between them, we compare the proteins with two state-of-the-art systems for structural classification of proteins: SCOP [19] and CATH [20]. Each system groups proteins into classes, or levels, based on shared structural characteristics. According to SCOP, 1D5TA and 1EKQA belong to the

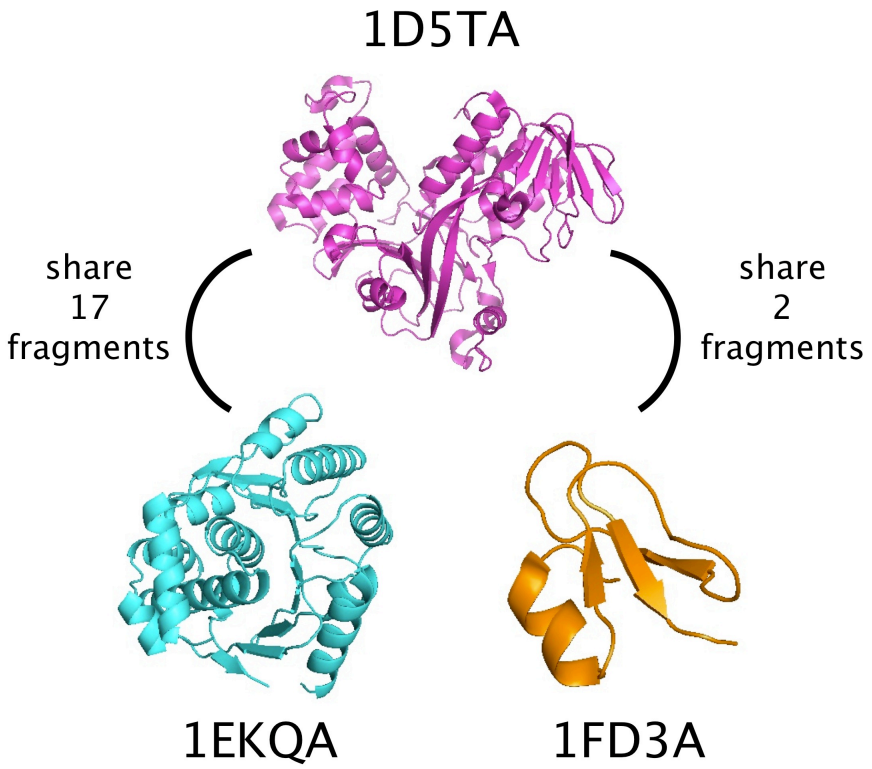


Figure 5.6: Comparing 1D5TA with two other proteins: 1EKQA (with which it shares 17 fragments), and 1FD3A (with which it shares 2 fragments). Protein structures are taken from the PDB [1].

same *Alpha and Beta* (a/b) class while 1FD3A is in a different *Small Proteins* class. CATH, on the other hand, classifies all three proteins as *Alpha Beta* proteins but while 1D5TA and 1EKQA have 3- and 2-Layer *Sandwich* folds respectively, 1FD3A has a different *Roll* fold.

Our results demonstrate that we can apply our framework to the analysis of the structure space of proteins. Given an alphabet of structural motifs, we can find all combinations of them that occur in a predefined number of protein structures. This information can further be used to generate graphs of the pairwise relationships between the proteins. In addition, for a given query protein, we can find all other proteins that share motifs with it and rank them according to how structurally similar

they are to the query protein based on the number of shared motifs. Note however, that if proteins A and B are structurally very similar but B is much smaller in size than A, then it is possible that a larger protein C that is less similar to A gets ranked higher than B in our ranking only because it shares more fragments with A. This problem arises from our definition of structural similarity and can be addressed by considering a different criteria for the oracle. As mentioned earlier, our current implementation does not take into account the relative location of the fragments in the proteins, and cannot capture multiple occurrences of the same fragment.

Note that storing only the maximal simplices allows us to build complexes of very large dimensions but prevents the direct access to non-maximal simplices. However, building more maximal complexes with varying N gives us an opportunity to extract all relevant information. Note that even though it took about 47 minutes to construct $K_1^{\mathcal{M}}$, we built $K_2^{\mathcal{M}}$ for less than 17 seconds and the runtime is expected to get even smaller as N increases.

Chapter 6

Conclusion

In this thesis, we present a novel computational approach for the construction of simplicial complexes over arbitrary topological spaces. The construction does not require a metric embedding space and allows us to build arbitrary simplicial complexes on various types of data. We present four algorithms for our oracle-based framework and compare two different but equivalent ways of representing the combinatorial structure in the computation. Our experiments indicate that the maximal representation of the simplicial complex is more efficient than its full representation and allows us to construct structures that we would otherwise not be able to construct due to limited computational resources.

In addition, we demonstrate the generality and utility of our framework by applying it to three different problems. In our Google application we construct simplicial complexes in order to find combinations of grocery items that result in a small number of recipes. The computational representations of the English, French and Bulgarian languages that we build, provide another example of the ability of our framework to construct models that capture certain structural properties among objects while creating opportunities for further topological analysis. Finally, by constructing simplicial complexes on different sets of structural motifs we can explore the similarity between protein structures as defined by the collection of their unique building blocks.

Note that some of the complexes that we build are very high dimensional, but we can construct them successfully with our maximal algorithms.

This project provides different directions for future work. One of them is to design a methodology for choosing an algorithm for the construction, which would involve approximating the sparsity and dimension of the complex. Another one would be performing a thorough theoretical analysis on the complexity of the four algorithms and further exploring the benefits and limitations of maximal versus full representation of the complexes. An immediate application of the latter analysis for instance, would be constructing simplicial complexes in parallel since the maximal simplices in a complex are not subsets of each other and can thus be built simultaneously. Furthermore, designing techniques for computing the homology of such maximal complexes without having to first enumerate all simplices, would allow for the topological analysis of very high dimensional complexes.

At the same time, the general definition of the oracle in our framework makes it possible to apply the presented modeling approach to different domains. Two examples include online shopping and library catalogue searches. Note that if the data is locally available, queries to the oracle will be much faster. Depending on the particular application we may not need to build the whole complex in order to obtain the desired information and in these cases our framework allows us to specify a stopping criteria for the construction. The existence of an optimal ordering of the vertices for Maximal-TD may also have interesting implications. Having found one such ordering by examining the behavior of the algorithm, we can retrieve it and further analyze its significance in the particular application.

Bibliography

[1] Research Collaboratory for Structural Bioinformatics PDB. `http://www.rcsb.org/pdb/`.

[2] D. Attali, A. Lieutier, and D. Salinas. Efficient data structure for representing and simplifying simplicial complexes in high dimension. In *Proc. 27th Ann. Sympos. Comput. Geom.*, June 13-15 2011.

[3] F. Bernstein, T. Koetzle, G. Williams, E. Meyer Jr., M. Brice, J. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. The Protein Data Bank: A computer-based archival file for macromolecular structures. *J. of. Mol. Biol.*, 112(1977):535.

[4] R. Bott and L. W. Tu. *Differential Forms in Algebraic Topology*. Springer-Verlag, New York, NY, 1982.

[5] G. Carlsson. Topology and data. *Bulletin of the American Mathematical Society (New Series)*, 46(2):255–308, 2009.

[6] F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407:564–568, 2008.

[7] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, New York, third edition, 2008.

[8] V. de Silva and G. Carlsson. Topological estimation using witness complexes. In *Proc. IEEE/Eurographics Symposium on Point-Based Graphics*, pages 157–166, 2004.

[9] D. Edelsbrunner, H. G. Kirkpatrick and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29:551–559, 1983.

[10] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13:43–72, 1994.

[11] K. Engel. *Sperner Theory*, volume 65. Cambridge University Press, New York, NY, 1997.

[12] J. Giesen and M. John. The flow complex: a data structure for geometric modeling. In *Proceedings of symposium on discrete algorithms*, pages 285–94, 2003.

[13] R. Graham, D. Knuth, and O. Patashnik. *Concrete mathematics: a foundation for computer science*. Addison-Wesley Publishing Company, Reading, MA, 1989.

[14] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, New York, NY, 2002.

[15] T. Kaczynski, K. Mischaikow, and M. Mrozek. *Computational Homology*. Springer-Verlag, New York, 2004.

[16] M. Kahle. Topology of random clique complexes. *Discrete Mathematics*, 309:1658–1671, 2009.

[17] D. Knuth. *The Art of Computer Programming*, volume 4. Addison-Wesley Professional, 2011.

[18] D. Kozlov. *Combinatorial Algebraic Topology*. Springer-Verlag, New York, NY, 2008.

[19] A. Murzin, S. Brenner, T. Hubbard, and C. Chothia. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *J. of. Mol. Biol.*, 247(4):536–540, 1995.

[20] C. Orengo, A. Michie, S. Jones, D. Jones, M. Swindells, and J. Thornton. CATH: A hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1108, 1997.

[21] J. Rossignac and P. Borrell. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455–465. Springer-Verlag, 1993.

[22] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of triangle meshes. *Proc. SIGGRAPH*, 26(2):65–70, 1992.

[23] L. Vietoris. Über den höheren Zusammenhang kompakter Räume und eine Klasse von zusammenhangstreuen Abbildungen. *Mathematische Annalen*, 97(1):454–472, 1927.

[24] A. Zomorodian. Fast construction of the Vietoris-Rips complex. *Computers & Graphics*, 34(3):263–271, 2010.

[25] A. Zomorodian. The tidy set: A minimal simplicial set for computing homology of clique complexes. In *Proc. ACM Symposium on Computational Geometry*, 2010.

[26] A. Zomorodian. *Advances in Applied and Computational Topology*, volume 70 of *Proceedings of Symposia in Applied Mathematics*, chapter Topological Data Analysis, pages 1–40. AMS, Providence, RI, 2012.

Appendix

object type	query term (grocery item)
$\boldsymbol{L}$	list of grocery items from left table in Figure 5.2
$\boldsymbol{b_i}$	i-th item from L
$\boldsymbol{g(b_i)}$	less: v_j where b_i is the j-th item in L sorted in incr. order of # hits
	more: v_j where b_i is the j-th item in L sorted in decr. order of # hits
$\boldsymbol{\mathcal{O}(\sigma)}$	query formed by "site:recipesindian.com" and $g^{-1}(v_i)$ for all $v_i \in \sigma$ returns nonzero number of results on Google search
total order	$<$
params	$n \in [1, 20]$
algs	MAXIMAL-BU MAXIMAL-TD with standard protocol
runs/pt	3 to 5

Table 1: Setup for Google Application

languages	English, French, Bulgarian
object type	letter from alphabet
$\boldsymbol{n}$	English: 26 French: 26 Bulgarian: 30
$\boldsymbol{b_i}$	i-th letter from alphabet
$\boldsymbol{g(b_i)}$	v_i
$\boldsymbol{D}$	Linux word list for the language English: 479,829 words French: 139,719 words Bulgarian: 823,209 words
$\boldsymbol{\mathfrak{O}(\sigma)}$	$\exists$ a word $w \in D$ such that $v_i \in \sigma \Rightarrow g^{-1}(v_i) \in w$ for all $v_i \in \sigma$
total order	$<$
algs	MAXIMAL-BU
runs/pt	2

Table 2: Setup for Languages Application

object type	structural motif
$\boldsymbol{n}$	60
$\boldsymbol{b_i}$	i-th motif
$\boldsymbol{c_i}$	number of tertiary contacts covered by b_i
$\boldsymbol{g(b_i)}$	less: v_j where b_i is the j-th motif in M sorted in incr. order of c
	more: v_j where b_i is the j-th motif in M sorted in decr. order of c
$\boldsymbol{\mathfrak{O}(\sigma)}$	$\exists$ at least N proteins, each of which contains $g^{-1}(v_i)$ for all $v_i \in \sigma$
total order	$<$
params	M – set of motifs on which complex is built N – lower bound on the number of proteins sharing a combination of the motifs
algs	MAXIMAL-BU MAXIMAL-TD with edge-dependent protocol
runs/pt	1

Table 3: Setup for Proteins Application