Computing Linking Numbers of a Filtration [*]

Herbert Edelsbrunner[†], and Afra Zomorodian[‡]

Abstract

We develop fast algorithms for computing the linking number of a simplicial complex within a filtration. We give experimental results in applying our work toward the detection of non-trivial tangling in biomolecules, modeled as alpha complexes.

Keywords. Computational geometry and topology, knots, linking number, three-manifolds, filtrations, alpha shapes, algorithms.

1 Introduction

In this paper, we develop fast algorithms for computing the linking numbers of simplicial complexes. Our work is within a framework of applying computational topology methods to the fields of biology and chemistry. Our goal is to develop useful tools by researchers in computational structural biology.

Motivation and Approach. In the 1980's, it was shown that the DNA, the molecular structure of the genetic code of all living organisms, can become knotted during replication [1]. This finding initiated interest in knot theory among biologists and chemists for the detection, synthesis, and analysis of knotted molecules [8]. The impetus for this research is that molecules with non-trivial topological attributes often display exotic chemistry. Taylor recently discovered a figure-of-eight knot in the structure of a plant protein by examining 3,440 proteins using a computer program [19]. Moreover, chemical self-assembly units have been used to create *catenanes*, chains of interlocking molecular rings, and *rotaxanes*, cyclic molecules threaded by linear molecules. Researchers are building nano-scale chemical switches and logic gates with these structures [2, 3]. Eventually, chemical computer memory systems could be built from these building blocks.

Catenanes and rotaxanes are examples of non-trivial structural tanglings. Our work is on detecting such interlocking structures in molecules through a combinatorial method, based on algebraic topology. We model biomolecules as a sequence of alpha complexes [7]. The basic assumption of this representation is that an alpha-complex sequence captures the topological features of a molecule. This sequence is also a filtration of the Delaunay triangulation, a well-studied combinatorial object, enabling the development of fast algorithms.

The focus of this paper is the linking number. Intuitively, this invariant detects if components of a complex are linked and cannot be separated. We hope to eventually incorporate our algorithm into publicly available software as a tool for detecting existence of interlocked molecular rings.

Given a filtration, the main contributions of this paper are:

(i) the extension of the definition of the linking number to graphs, using a canonical basis,

(ii) an algorithm for enumerating and generating all cycles and their spanning surfaces within a filtration,

(iii) data structures for efficient enumeration of co-existing pairs of cycles in different components,

(iv) an algorithm for computing the linking number of a pair of cycles,

(v) and the implementation of the algorithms and experimentation on real data sets.

Algorithm (iv) is based on spanning surfaces of cycles, giving us an approximation to the linking number in the case of non-orientable or self-intersecting surfaces. Such cases do not arise often in practice, as shown in Section 6. However, we note in Section 2 that the linking number of a pair may be also computed by alternate algorithms. Regardless of the approach taken, pairs of potentially linked cycles must be first detected and enumerated. We provide the algorithms and data structures of such enumeration in (i-iii).

[*]Research by both authors is partially supported by ARO under grant DAAG55-98-1-0177. Research by the first author is also partially supported by NSF under grant CCR-97-12088, EIA-99-72879, and CCR-00-86013.

[†]Department of Computer Science, Duke University, Durham, and Raindrop Geomagic, Research Triangle Park, NC.

[‡]Department of Computer Science, University of Illinois, Urbana, IL.

Prior work. Important knot problems were shown to be decidable by Haken in his seminal work on normal surfaces [10]. This approach, as reformulated by Jaco and others [13], forms the basis of many current knot detection algorithms. Haas et al. recently showed that these algorithms take exponential time in the number of crossings in a knot diagram [12]. They also placed both the UNKNOTTING PROBLEM and the SPLITTING PROBLEM in NP, the latter being the focus of our paper. Generally, other approaches to knot problems have unknown complexity bounds, and are assumed to take at least exponential time. As such, the state of the art in knot detection only allows for very small data sets. We refer to Adams [1] background in knot theory.

Three-dimensional alpha shapes and complexes may be found in Edelsbrunner and Mücke [7]. We modify the persistent homology algorithm to compute cycles and surfaces [6]. We refer to Munkres [15] for background in homology theory that is accessible to non-specialists.

Outline. The remainder of this paper is organized as follows. We review linking numbers for collections of closed curves, and extend this notion to graphs in $\mathbb{R}^3$ in Section 2. We describe our model for molecules in Section 3. Extending the persistence algorithm, we design basic algorithms in Section 4 and use them to develop an algorithm for computing linking numbers in Section 5. We show results of some initial experiments in Section 6, concluding the paper in Section 7.

2 Linking Number

In this section, we define links and discuss two equivalent definitions of the linking number. While the first definition provides intuition, the second definition is the basis of our computational approach.

Links. A *knot* is an embedding of a circle in three-dimensional Euclidean space, $k : \mathbb{S}^1 \to \mathbb{R}^3$. Two knots are *equivalent* if there is an ambient isotopy that maps the first to the second. That is, we may deform the first to the second by a continuous motion that does not cause self-intersections. A *link* l is a collection of knots with disjoint images. A link is *separable (splitable)* if it can be continuously deformed so that one or more components can be separated from other components by a plane that itself does not intersect any of the components. We often visualize a link l by a *link diagram*, which is the projection of a link onto a plane such that the over- and under-crossings of knots are presented clearly, We give an example in Figure 1. For a formal definition, see [12].

Linking number. A *knot (link) invariant* is a function that assigns equivalent objects to equivalent knots (links.) Seifert first defined an integer link invariant, the linking number,

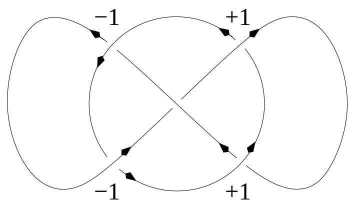

Figure 1: A link diagram for the Whitehead link. Vertices occur at crossings and are labeled according to the convention in Figure 2.

in 1935 to detect link separability [18]. Given a link diagram for a link l, we choose orientations for each knot in l. We then assign integer labels to each crossing between any pair of knots k, k', following the convention in Figure 2. Let

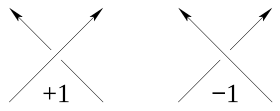

Figure 2: The crossing label is $+1$ if the rotation of the overpass by 90 degrees counter-clockwise aligns its direction with the underpass, and -1 otherwise.

$\lambda(k, k')$ of the pair of knots to be one half the sum of these labels. A standard argument using Reidermeister moves shows that λ is an invariant for equivalent pairs of knots up to sign [1]. The *linking number* $\lambda(l)$ of a link l is

$$\lambda(l) \quad = \quad \sum_{k \neq k' \in l} |\lambda(k, k')|.$$

We note that $\lambda(l)$ is independent of knot orientations. Also, the linking number does not completely recognize linking. The Whitehead link in Figure 1, for example, has linking number zero, but is not separable.

Surfaces. The linking number may be equivalently defined by other methods, including one based on surfaces [17]. A *spanning surface* for a knot k is an embedded surface with boundary k. An orientable spanning surface is a *Seifert surface*. Because it is orientable, we may label its two sides as positive and negative. We show examples of such surfaces for the Hopf link in Figure 3.

Given a pair of oriented knots k, k', and a Seifert surface s for k, we label s by using the orientation of k. We then adjust k' via a homotopy h until it meets s in a finite number of points. Following along k' according to its orientation, we add $+1$ whenever k' passes from the negative to the positive side, and -1 whenever k' passes from the positive to the negative side. The following lemma asserts that this sum is independent of our the choice of h and s, and it is, in fact, the linking number.

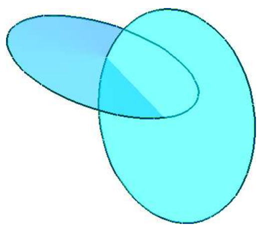

Figure 3: The Hopf link and Seifert surfaces of its two unknots. Clearly, $\lambda = 1$. This link is the 200th complex for data set H in Section 6.

SEIFERT SURFACE LEMMA. $\lambda(k, k')$ is the sum of the signed intersections between k' and any Seifert surface for k.

The proof is by a standard Seifert surface construction [17]. If the spanning surface is non-orientable, we can still count how many times we pass through the surface, giving us the following weaker result.

SPANNING SURFACE LEMMA. $\lambda(k, k') \pmod 2$ is the parity of the number of times k' passes through any spanning surface for k.

Graphs. We need to extend the linking number to graphs, in order to use the above lemma for computing linking numbers for simplicial complexes. Let $G = (V, E), E \subseteq \binom{V}{2}$ be a simple undirected graph in $\mathbb{R}^3$ with c components G^i. Let $z_1, \ldots, z_m$ be a fixed basis for the cycles in G, where $m = |E| - |V| + c$. We then define the linking number between two components of G to be $\lambda(G^i, G^j) = |\lambda(z_p, z_q)|$ for all cycles z_p, z_q in G^i, G^j, respectively. The linking number of G is then defined by combining the total interaction between pairs of components:

$$\lambda(G) = \sum_{i \neq j} \lambda(G^i, G^j).$$

The linking number is computed only between pairs of components following Seifert's original definition. Linked cycles within the same component may be easily unlinked by a homotopy. Figure 4 shows that the linking number for graphs is dependent on the chosen basis. While it may seem that we want $\lambda(G) = 1$ in the figure, there is no clear answer in general. We will define a canonical basis in Section 4 using the persistent homology algorithm to compute $\lambda(G)$ for simplicial complexes.

3 Alpha Complexes

Our approach to analyzing a topological space is to assume a filtration for such a space. A filtration may be viewed as a history of a growing space that is undergoing geometric and topological changes. While filtrations may be obtained by various methods, only meaningful filtrations give meaningful linking numbers. As such, we use alpha complex filtrations to model molecules. The alpha complex captures the connectivity of a molecule that is represented by a union of spheres. This model may be viewed as the dual of the space filling model for molecules [14].

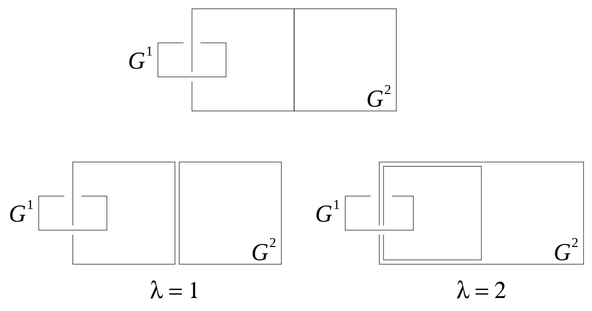

Figure 4: We get different $\lambda(G)$ for graph G (top) depending on our choice of basis for G^2: two small cycles (left) or one large and one small cycle (right.)

Dual complex. A *spherical ball* $\hat{u} = (u, U^2) \in \mathbb{R}^3 \times \mathbb{R}$ is defined by its center u and square radius U^2. If $U^2 < 0$, the radius is imaginary and so is the ball. The *weighted distance* of a point x from a ball $\hat{u}$ is $\pi_{\hat{u}}(x) = \|x - u\|^2 - U^2$. Note that a point $x \in \mathbb{R}^3$ belongs to the ball iff $\pi_{\hat{u}}(x) \leq 0$, and it belongs to the bounding sphere iff $\pi_{\hat{u}}(x) = 0$. Let S be a finite set of balls. The *Voronoi region* of $\hat{u} \in S$ is the set of points for which $\hat{u}$ minimizes the weighted distance,

$$V_{\hat{u}} = \{x \in \mathbb{R}^3 \mid \pi_{\hat{u}}(x) \leq \pi_{\hat{v}}(x), \forall \hat{v} \in S\}.$$

The Voronoi regions decompose the union of balls into convex cells of the form $\hat{u} \cap V_{\hat{u}}$, as illustrated in Figure 5. Any

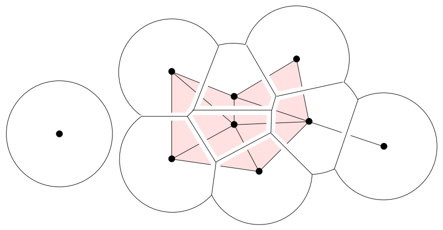

Figure 5: Union of nine disks, convex decomposition using Voronoi regions, and dual complex.

two regions are either disjoint or they overlap along a shared portion of their boundary. We assume general position, where at most four Voronoi regions can have a non-empty common intersection. Let $T \subseteq S$ have the property that its Voronoi regions have a non-empty common intersection, and consider the convex hull of the corresponding centers, $\sigma_T = \operatorname{conv}\{u \mid \hat{u} \in T\}$. General position implies that σ_T is a d-dimensional simplex, where $d = \operatorname{card} T - 1$. The *dual complex* of S is the collection of simplices constructed in this manner,

$$K = \{\sigma_T \mid T \subseteq S, \bigcap_{\hat{u} \in T} (\hat{u} \cap V_{\hat{u}}) \neq \emptyset\}.$$

Any two simplices in K are either disjoint or they intersect in a common face which is a simplex of smaller dimension. Furthermore, if $\sigma \in K$, then all faces of σ are simplices in K. A set of simplices with these two properties is a *simplicial complex* [15]. A *subcomplex* is a subset $L \subseteq K$ that is itself a simplicial complex.

Alpha complex. A *filtration ordering* is an ordering of a set of simplices such that each prefix of the ordering is a subcomplex. The sequence of subcomplexes defined by taking successively larger prefixes is the corresponding *filtration*. For dual complexes of a collection of balls, we generate an ordering and a filtration by literally growing the balls. For every real number $\alpha^2 \in \mathbb{R}$, we increase the square radius of a ball $\hat{u}$ by α^2, giving us $\hat{u}(\alpha) = (u, U^2 + \alpha^2)$. We denote the collection of expanded balls $\hat{u}(\alpha)$ as $S(\alpha)$. If $U^2 = 0$, then α is the radius of $\hat{u}(\alpha)$. If $\alpha^2 < 0$, then α is imaginary, and so is the ball $\hat{u}(\alpha)$. The α*-complex* $K(\alpha)$ of S is the dual complex of $S(\alpha)$ [7]. For example, $K(-\infty) = \emptyset$, $K(0) = K$, and $K(\infty) = D$ is the dual of the Voronoi diagram, also known as the Delaunay triangulation of S. For each simplex $\sigma \in D$, there is a unique *birth time* $\alpha^2(\sigma)$ defined such that $\sigma \in K(\alpha)$ iff $\alpha^2 \geq \alpha^2(\sigma)$. We order the simplices such that $\alpha^2(\sigma) < \alpha^2(\tau)$ implies σ precedes τ in the ordering. More than one simplex may be born at a time and such cases may arise even if S is in general position. In the case of a tie, it is convenient to order lower-dimensional simplices before higher-dimensional ones, breaking remaining ties arbitrarily. We call the resulting sequence the *age ordering* of the Delaunay triangulation.

Modeling molecules. To model molecules by alpha complexes, we use representations of molecules as unions of balls. Each ball is an atom, as defined by its position in space and its van der Waals radius. These atoms become the spherical balls we need to define our complexes. Our representation gives us a filtration of alpha complexes for each molecule, as shown in Figure 6. We compute a linking number for each complex in a filtration of m complexes. Let $[m]$ denote the set $\{1, 2, \ldots, m\}$. Then, the linking number may be viewed as a *signature function* $\lambda : [m] \to \mathbb{Z}$ that maps each index $i \in [m]$ to an integer $\lambda(i) \in \mathbb{Z}$. For other signature functions for filtrations of alpha complexes, see [5, 7].

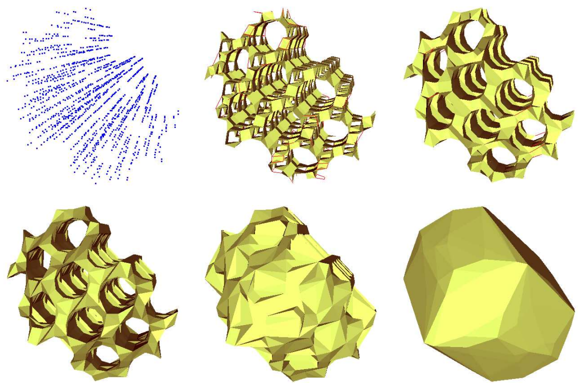
Figure 6: Six complexes in the filtration of 42,787 complexes for data set Z in Section 6.

4 Basis and Surfaces

To compute the linking numbers for an alpha complex, we need to recognize cycles, establish a basis for the set of cycles, and find spanning surfaces for the basis cycles. We do so by extending an algorithm we developed for computing persistent homology [6]. We dispense with defining persistence and concentrate on the algorithm and its extension.

Homology. We use homology to define cycles in a complex. Homology partitions cycles into equivalence classes using the boundary class of bounding cycles as the null element of a quotient group in each dimension. We use $\mathbb{Z}_2$ homology, so the group operation, which we call *addition*, is symmetric difference. Addition allows us to combine sets of simplices in a way that eliminates shared boundaries, as shown in Figure 7. Intuitively, non-bounding 1-cycles correspond to the graph notion of a cycle. We need to define a basis for the first homology group of the complex which contains all 1-cycles, and choose representatives for each homology class. We use these representatives to compute linking numbers for the complex.

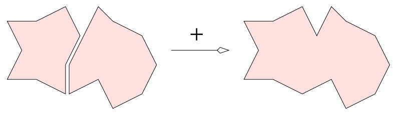

Figure 7: Symmetric difference in dimensions one and two. We add two 1-cycles to get a new 1-cycle. We add the surfaces the cycles bound to get a spanning surface for the new 1-cycle.

A simplex of dimension d in a filtration either creates a d-cycle or destroys a $(d-1)$-cycle by turning it into a boundary. We mark simplices as *positive* or *negative*, according to this action [5]. In particular, edges in a filtration which connect components are marked as negative. The set of all negative edges gives us a spanning tree of the complex, as shown in Figure 8. We use this spanning tree to define our canonical basis. Every time a positive edge σ_i is added to the complex, it creates a new cycle. We choose the unique cycle that contains σ_i and no other positive edge as a new basis cycle. We call this cycle a *canonical cycle*, and the collection of canonical cycles, the *canonical basis*. We use this basis for computation.

Persistence. The persistence algorithm matches positive and negative simplices to find life-times of homological cycles in

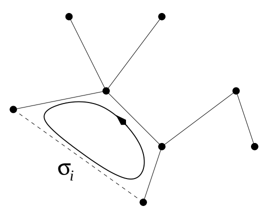

Figure 8: Solid negative edges combine to form a spanning tree. The dashed positive edge σ_i creates a canonical cycle.

a filtration. The algorithm does so by following a representative cycle z for each class. Initially, z is the boundary of a negative simplex σ_j, as z must lie in the homology class σ_j destroys. The algorithm then successively adds class-preserving boundary cycles to z until it finds the matching positive simplex σ_i, as shown in Figure 9. We call the half-

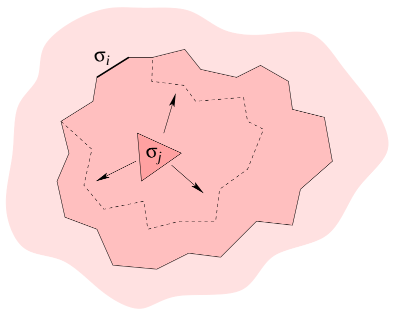

Figure 9: Starting from the boundary of the negative triangle σ_j, the persistence algorithm finds a matching positive edge σ_i by finding the dashed 1-cycle. We modify this 1-cycle further to find the solid canonical 1-cycle and a spanning surface.

open interval $[i, j)$ the *persistence interval* of both the homology class and its canonical representative. During this interval, the homology class exists as a class of homologous non-boundings cycles in the filtration. As such, the class may only affect the linking numbers of complexes $K_i, \ldots, K_{j-1}$ in the filtration. We use this insight in the next section to design an algorithm for computing linking numbers.

Computing canonical cycles. The persistence algorithm halts when it finds the matching positive simplex σ_i for a negative simplex σ_j, often generating a cycle z with multiple positive edges and multiple components. We need to convert z into a canonical cycle by eliminating all positive edges in z except for σ_i. We call this process *canonization*. To canonize a cycle, we add cycles associated with unnecessary positive edges to z successively, until z is composed of σ_i and negative edges, as shown in Figure 9. Canonization amounts to replacing one homology basis element with a linear combination of other elements in order to reach the unique canonical basis we defined earlier. A cycle undergoing canonization changes homology classes, but the rank of the basis never changes.

Computing spanning surfaces. For each canonical cycle, we need a spanning surface in order to compute linking numbers. We may compute these by maintaining surfaces while computing the cycles. Recall that initially, a cycle representative is the boundary of a negative simplex σ_j. We use σ_j as the initial spanning surface for z. Every time we add a cycle y to z in the persistence algorithm, we also add the surface y bounds to the z's surface. We continue this process through canonization to produce both canonical cycles and their spanning surfaces. Here, we are using a crucial property of our filtrations: the final complex is always the Delaunay complex of the set of weighted points and does not contain any 1-cycles. Therefore, all 1-cycles are eventually turned to boundaries and have spanning surfaces.

If the generated spanning surface is Seifert, we may apply the SEIFERT SURFACE LEMMA to compute the linking numbers. In some cases, however, the spanning surface is not Seifert, as in Figure 10. In these cases, we may either compute the linking number modulus 2 by applying the SPANNING SURFACE LEMMA, or compute the linking number by alternative methods.

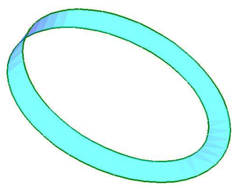

Figure 10: The spanning surface produced for the cycle which is the boundary of a Möbius strip is non-orientable.

5 Algorithm

In this section, we use the basis and spanning surfaces computed for 1-cycles to find linking numbers for all complexes in a filtration. Since we focus on 1-cycles only, we will refer to them simply as cycles.

Overview. We assume a filtration $K_1, K_2, \ldots, K_m$ as input, which we alternately view as a single complex undergoing growth. As simplices are added, the complex undergoes topological changes which affect the linking number: new components are created and merged together, and new non-bounding cycles are created and eventually destroyed. We use a basic insight from the last section: a basis cycle z with persistence interval $[i, j)$ may only affect the linking numbers of complexes $K_i, K_{i+1}, \ldots, K_{j-1}$ in the filtration, Consequently, we only need to consider basis cycles z' that exist during some subinterval $[u, v) \subseteq [i, j)$ in a different component than z's. We call the pair z, z' a *potentially-linked (p-linked) pair* of basis cycles, and the interval $[u, v)$ the *p-linking interval*.

Focusing on p-linked pairs, we get an algorithm with three phases. In the first phase, we compute all p-linked pairs of cycles. In the second phase, as shown in Figure 11, we compute the linking numbers of such pairs. In the third and final phase, we aggregate these contributions to find the linking number signature for the filtration.

```
for each p-linked pair $z_p, z_q$ with interval $[u, v)$ do
  Compute $\lambda = |\lambda(z_p, z_q)|$ ;
  Output $(\lambda, [u, v))$
endfor.
```

Figure 11: Linking number algorithm.

Two cycles z_p, z_q with persistence intervals $[i_p, j_p)$,$[i_q, j_q)$ co-exist during $[r, s) = [i_p, j_p) \cap [i_q, j_q)$. We need to know if these cycles also belong to different components during some sub-interval $[u, v) \subseteq [r, s)$. Let $t_{p,q}$ be the minimum index in the filtration when z_p and z_q are in the same component. Then, $[u, v) = [r, s) \cap [0, t_{p,q})$. If $[u, v) \neq \emptyset$, z_p, z_q are p-linked during that interval. In the remainder of this section, we will first develop a data structure for computing $t_{p,q}$ for any pair of cycles z_p, z_q. Then, we use this data structure to efficiently enumerate all pairs of p-linked cycles. Finally, we give an algorithm for computing $\lambda(z_p, z_q)$ for a p-linked pair of cycles z_p, z_q.

Component history. To compute $t_{p,q}$, we need to have a history of the changes to the set of components in a filtration. There are two types of simplices that can change this set. Vertices create components and are therefore all positive. Negative edges connect components. We construct a binary tree called *component tree* recording these changes using a union-find data structure [4]. The leaves of the component tree are the vertices of the filtration. When a negative edge connects two components, we create an internal node and connect it to the nodes representing these components, as shown in Figure 12. The component tree has size $O(n)$ for

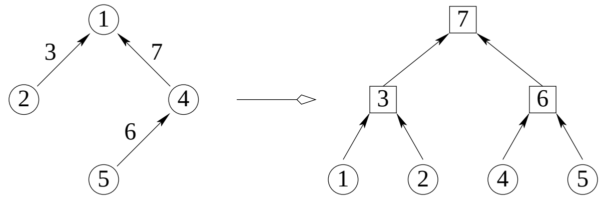

Figure 12: The union-find data structure (left) has vertices as nodes and negative edges as edges. The component tree (right) has vertices as leaves and negative edges as internal nodes.

n vertices, and we construct it in time $O(nA^{-1}(n))$, where $A^{-1}(n)$ is the inverse of the Ackermann's function which exhibits insanely slow growth. Having constructed the component tree, we find the time two vertices w, x are in the same component by finding their lowest common ancestor (lca) in this tree. We utilize Harel and Tarjan's optimal method to find lca's with $O(n)$ preprocessing time and $O(1)$ query time [11]. Their method uses bit operations. If such operations are not allowed, we may use van Leeuwen's method with the same preprocessing time and $O(\log \log n)$ query time [20].

Enumeration. Having constructed the component tree, we use a modified union-find data structure to enumerate all pairs of p-linked cycles. We augment the data structure to allow for quick listing of all existing canonical cycles in each component in K_i. Our augmentation takes two forms: we put the roots of the disjoint trees, representing components, into a circular doubly-linked list. We also store all existing cycles in each component in a doubly-linked list at the root node of the component, as shown in Figure 13. When com-

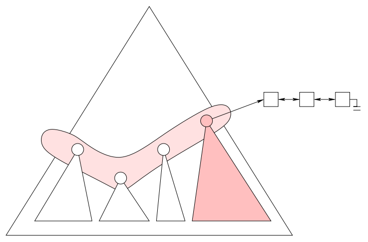

Figure 13: The augmented union-find data structure places root nodes in the shaded circular doubly-linked list. Each root node stores all active canonical cycles in that component in a doubly-linked list, as shown for the darker component.

ponents merge, the root x_1 of one component becomes the parent of the root x_2 of the other component. We concatenate the lists stored at the x_1, x_2, store the resulting list at x_1, and eliminate x_2 from the circular list in $O(1)$ time. When cycle z_p is created at time i, we first find z_p's component in time $O(A^{-1}(n))$. Then, we store z_p at the root of the component and keep a pointer to z_p with simplex σ_j, which destroys z_p. This implies that we may delete z_p from the data structure at time j with constant cost.

Our algorithm to enumerate p-linked cycles is incremental. We add and delete cycles using the above operations from the union-find forest, as the cycles are created and deleted in the filtration. When a cycle z_p is created at time i, we output all p-linked pairs in which z_p participates. We start at the root which now stores z_p and walk around the circular list of roots. At each root x, we query the component tree we constructed in the last subsection to find the time t when the component of x merges with that of z_p. Note that $t = t_{p,q}$ for all cycles z_q stored at x. Consequently, we can compute the p-linking interval for each pair z_p, z_q to determine if the pair is p-linked. If the filtration contains P p-linked pairs, our algorithm takes time $O(mA^{-1}(n) + P)$, as there are at most m cycles in the filtration.

Orientation. In the previous section, we showed how one may compute spanning surfaces s_p, s_q for cycles z_p, z_q, respectively. To compute the linking number using our lemma, we need to orient either the pair s_p, z_q or z_p, s_q. Orienting a cycle is trivial: we orient one edge and walk around to orient the cycle. If either surface has no self-intersections, we may easily attempt to orient it by choosing an orientation for an arbitrary triangle on the surface, and spreading that orientation throughout. The procedure either orients the surface or classifies it as non-orientable. We currently do not have an algorithm for orienting surfaces with self-intersections. The main difficulty is distinguishing between two cases for a self-intersection: a surface touching itself and passing through itself, as shown in Figure 14.

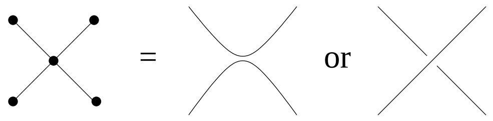

Figure 14: A surface self-intersection viewed from its side. We cannot resolve it as the surface touching or passing through itself.

Computing $\boldsymbol{\lambda}$. We now show how to compute $\lambda(z_p, z_q)$ for a pair of p-linked cycles z_p, z_q, completing the description of our algorithm in Figure 11. We assume that we have oriented s_p, z_q for the remainder of this subsection.

Let the *star* of a vertex u $\mathrm{St}\, u$ be the set of simplices containing u as a vertex. We subdivide the complex via a *barycentric subdivision* by connecting the centroid of each triangle to its vertices and midpoints of its edges, subdividing the simplices accordingly. This subdivision guarantees that no edge uv will have both ends on a Seifert surface unless it is entirely contained in that surface. We note that this approach mimics the construction of regular neighborhoods for complexes [9].

For a vertex $u \in s_p$, the edge property guaranteed by subdivision enables us to mark each edge $uv \in \mathrm{St}\, u, v \notin s_p$ as positive or negative, depending on the location of v with respect to s_p. We show an example of this marking in Figure 15. After marking edges, we walk once around z_q, starting at a vertex not on s_p. If such a vertex does not exist, then $\lambda(z_p, z_q) = 0$. Otherwise, we create a string $S_{p,q}$ of $+$ and $-$ characters by noting the marking of edges during our walk. $S_{p,q}$ has even length as we start and end our walk on a vertex not on s_p, and each intersection of z_q with s_p produces a pair of characters, as shown in Figure 16. If $S_{p,q}$ is the empty string, z_q never intersects s_p and $\lambda(z_p, z_q) = 0$. Otherwise, z_q passes through s_p for pairs $+-$ and $-+$, corresponding to z_q piercing the positive or negative side of

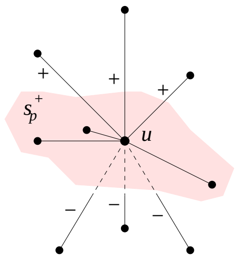

Figure 15: Edges $uv \in \mathrm{St}\, u, u \in s_p, v \notin s_p$ are marked $+$ or $-$ depending on where they end relative to the oriented Seifert surface s_p.

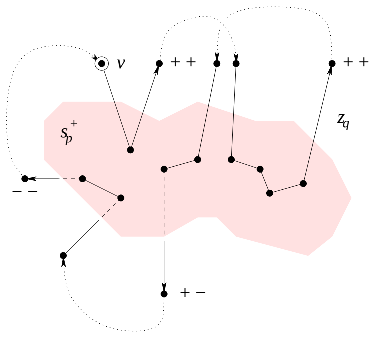

Figure 16: Starting at v, we walk on z_q according to its orientation. Segments of z_q that intersect s_p are shown, along with their contribution to $S_{p,q} =$ "$++++++----$". We get $\lambda(z_p, z_q) = -1$.

s_p, respectively. Scanning $S_{p,q}$ from left to right in pairs, we add $+1$ for each occurrence of $-+$, -1 for each $+-$, and 0, for each $++$ or $--$. Applying the SEIFERT SURFACE LEMMA in Section 2, we see that this sum is $\lambda(z_p, z_q)$.

Computing $\boldsymbol{\lambda} \bmod 2$. If neither of the spanning surfaces s_p, s_q of the two cycles z_1, z_2 is Seifert, we may still compute $\lambda(z_1, z_2) \bmod 2$ by a modified algorithm, provided one surface, say s_p, has no self-intersections. We choose an orientation on s_p locally, and extend it until all the stars of the original vertices are oriented. are oriented. This orientation will not be consistent globally, resulting in pair of adjacent vertices in s_p with opposite orientations. We call the implicit boundary between vertices with opposite orientations a *flip curve*, as shown in bold in Figure 17. When a cycle segment crosses the flip curve, orientation changes. Therefore, in addition to noting marked edges, we add a $+$ to the string $S_{p,q}$ every time we cross a flip line. To compute $\lambda(z_p, z_q) \bmod 2$, we only count $+$'s in $S_{p,q}$ and take the parity as our answer.

If s_p is orientable, there are no flip curves on it. The contribution of cycle segments to the string is the same as before: $+-$ or $-+$ for segments that pass through s_p, and $++$ and $--$ for segments that do not. By counting $+$'s, only segments that pass through s_p change the parity of the sum for λ. Therefore, the algorithm computes $\lambda \bmod 2$ correctly

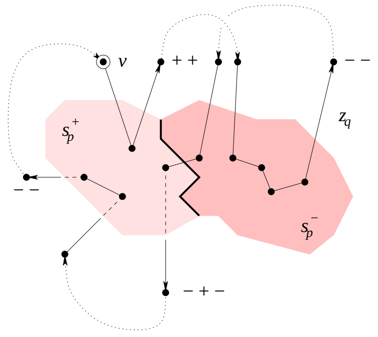

Figure 17: The bold flip curve is the border of s_p^+ and s_p^-, the portions of s_p that are oriented differently. $S_{p,q}$ = "$++--+---$", so counting all $+$'s, we get $\lambda(z_p, z_q) \bmod 2 = 3 \bmod 2 = 1$.

for orientable surfaces. For the orientable surface on the right in Figure 16, for instance, we get $\lambda(z_p, z_q) \bmod 2 = 5 \bmod 2 = 1$, which is equivalent to the parity of the answer computed by the previous algorithm.

Remark. We are currently examining the question of orienting surfaces with self-intersections. Using our current methods, we may obtain a lower bound signature for λ by computing a mixed sum: we compute λ and $\lambda \bmod 2$ whenever we can to obtain the approximation. We may also develop other methods, including those based on the projection definition of the linking number in Section 2.

6 Experiments

In this section, we present some experimental timing results and statistics which we used to guide our algorithm development. We also provide visualizations of basis cycles in a filtration. All timings were done on a Micron PC with a 266 MHz Pentium II processor and 128 MB RAM running Solaris 8.

Implementation. We have implemented all the algorithms in the paper, except for the algorithm for computing $\lambda \bmod 2$. Our implementation differs from our exposition in three ways. The implemented component tree is a standard union-find data structure with the union by rank heuristic, but no path compression [4]. Although this structure has a $O(n \log n)$ construction time and a $O(\log n)$ query time, it is simple to implement and extremely fast in practice. We also use a heuristic to reduce the number of p-linked cycles. We store bounding boxes at the roots of the augmented union-find data structure. Before enumerating p-linked cycles, we check to see if the bounding box of the new cycle intersects with that of the stored cycles. If not, the cycles cannot be linked, so we obviate their enumeration. Finally, we only simulate the barycentric subdivision.

Data. We have experimented with a variety of data sets and show the results for six representative sets in this section. The first data set contains points regularly sampled along two linked circles. The resulting filtration contains a complex which is a Hopf link, as shown in Figure 3. The other data sets represent molecular structures with weighted points. In each case, we first compute the weighted Delaunay triangulation and the age ordering of that triangulation. The data points become vertices or 0-simplices. Table 1 gives the sizes of the data sets, their Delaunay triangulations, and age orderings. We show renderings of specific complexes in the filtration for data set K in Figure 18.

	# simplices of dimension d				total
	0	1	2	3	
H	100	1,752	3,240	1,587	6,679
G	318	2,322	3,978	1,973	8,591
M	1,001	7,537	13,018	6,481	28,037
Z	1,296	11,401	20,098	9,992	42,787
K	2,370	17,976	31,135	15,528	67,009
D	7,774	60,675	105,710	52,808	226,967

Table 1: H defines a Hopf link. G is Gramicidin A, a small protein. M is a protein monomer. Z is a portion of a periodic zeolite structure. K is a human cyclin-dependent kinase. D is a DNA tile.

Basis. Table 2 summarizes the basis generation process. We distinguish the two steps of our algorithm: initial basis generation and canonization. We give the number of basis cycles for the entire filtration, which is equal to the number of positive edges. We show the effect of canonization on

	time in seconds			# cycles
	generate	canonize	total	
H	0.08	0.04	0.12	1,653
G	0.08	0.03	0.11	2,005
M	0.28	0.20	0.48	6,537
Z	0.46	0.46	0.92	10,106
K	0.72	1.01	1.73	15,607
D	2.63	2.94	5.57	52,902

Table 2: Time to generate and canonize basis cycles, as well as their number.

the size of the cycles and their spanning surfaces in Table 3. Note that canonization increases the size of cycles by one or two orders of magnitude. This is partially the reason we try to avoid performing the link detection if possible.

Links. In Table 4, we show that our component tree and augmented trees are very fast in practice to generate p-linked

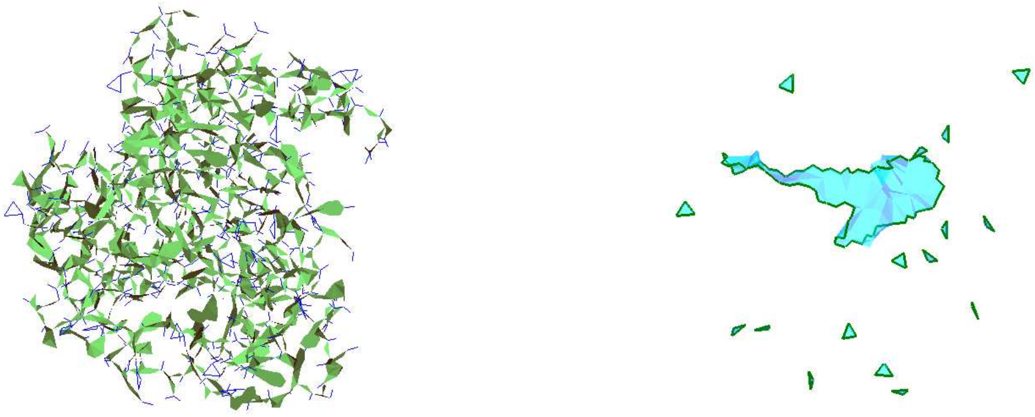
Figure 18: Complex K_{8168} of K has two components and seventeen cycles. The spanning surfaces are rendered transparently.

	avg cycle length		avg surface size	
	before	after	before	after
H	3.06	51.03	1.06	63.04
G	3.26	13.02	1.38	52.28
M	3.29	34.18	1.33	71.18
Z	4.71	25.33	3.26	117.81
K	3.48	67.87	1.62	166.70
D	3.46	39.94	1.81	158.99

Table 3: Average number of edges per cycle and number of triangles per spanning surface, before and after canonization.

pairs. We also show that our bounding box heuristic for reducing the number of p-linked pairs increases the computation time negligibly. The heuristic is quite successful, more-

	tree	alg	heur	links
H	0.01	0.00	0.00	0.01
G	0.00	0.01	0.02	0.02
M	0.03	0.06	0.06	0.23
Z	0.04	0.07	0.07	0.13
K	0.06	0.13	0.16	0.36
D	0.27	0.56	0.82	8.22

Table 4: Time in seconds to construct the component tree, and enumerate p-linked pairs (alg), p-linked pairs with intersecting bounding boxes (heur), and links.

over, in reducing the number of pairs we have to check for linkage It eliminates 99.8% of the candidates for dataset Z, for example, as shown in Table 5. The differences in total time of computation reflect the basic structure of the datasets. Dataset D has a large computation time, for instance, as the average size of the p-linked surfaces is approximately 264.16 triangles, compared to about 1.88 triangles for dataset K, and about 1.73 triangles for dataset M.

	alg	heur	links
H	1	1	1
G	112	0	0
M	16,503	14,968	0
Z	169,594	308	0
K	12,454	11,365	0
D	98,522	4,448	0

Table 5: Number of p-linked pairs (alg), p-linked pairs with intersecting bounding boxes (heur), and links.

Discussion. Our initial experiments demonstrate the feasibility of the algorithms for fast computation of linking. The experiments fail to detect any links in the protein data, however. This is to be expected, as a protein consists of a single component, the primary structure of a protein being a single polypeptide chain of amino acids. Links, on the other hand, exist in different components by definition. We may relax this definition easily, however, to allow for links occuring in the same component. We have implementations of algorithms corresponding to this relaxed definition. Our future plans include looking for links in proteins from the Protein Data Bank [16]. Such links could occur naturally as a result of disulphide bonds between different residues in a protein.

7 Conclusion

In this paper, we develop algorithms for finding the linking numbers of a filtration. We give algorithms for computing bases of 1-cycles and their spanning surfaces in simplicial complexes, and enumerating co-existing cycles in different components. In addition, we present an algorithm for computing the linking number of a pair of cycles using the surface formulation. Our implementations show that the algorithms are fast and feasible in practice. By modeling molecules as filtrations of alpha complexes, we can detect potential non-trivial tangling within molecules. Our work is within a framework for applying topological methods for understanding molecular structures.

Acknowledgments. We thank David Letscher for discussions during the early stages of this work. We also thank Daniel Huson for the zeolite dataset Z, and Thomas La Bean for the DNA tile data set D.

References

[1] ADAMS, C. C. *The Knot Book: An Elementary Introduction to the Mathematical Theory of Knots*. W. H. Freeman and Company, New York, NY, 1994.

[2] BISSELL, R. A., CÓRDOVA, E., KAIFER, A. E., AND STODDART, J. F. A checmically and electrochemically switchable molecular shuttle. *Nature* **369** (1994), 133–137.

[3] COLLIER, C. P., WONG, E. W., BELOHRADSKÝ, RAYMO, F. M., STODDART, J. F., KUEKES, P. J., WILLIAMS, R. S., AND HEATH, J. R. Electronically configurable moleculear-based logic gates. *Science* **285** (1999), 391–394.

[4] CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1994.

[5] DELFINADO, C. J. A., AND EDELSBRUNNER, H. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Comput. Aided Geom. Design* **12** (1995), 771–784.

[6] EDELSBRUNNER, H., LETSCHER, D., AND ZOMORODIAN, A. Topological persistence and simplification. In *Proc. 41st Ann. IEEE Sympos. Found. Comput. Sci.* (2000), pp. 454–463.

[7] EDELSBRUNNER, H., AND MÜCKE, E. P. Three-dimensional alpha shapes. *ACM Trans. Graphics* **13** (1994), 43–72.

[8] FLAPAN, E. *When Topology Meets Chemistry : A Topological Look at Molecular Chirality*. Cambridge University Press, New York, NY, 2000.

[9] GIBLIN, P. J. *Graphs, Surfaces, and Homology*, second ed. Chapman and Hall, New York, NY, 1981.

[10] HAKEN, W. Theorie der Normalfl'achen. *Acta Math.* **105** (1961), 245–375.

[11] HAREL, D., AND TARJAN, R. E. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* **13** (1984), 338–355.

[12] HASS, J., LAGARIAS, J. C., AND PIPPENGER, N. The computational complexity of knot and link problems. *J. ACM* **46** (1999), 185–211.

[13] JACO, W., AND TOLLEFSON, J. L. Algorithms for the complete decomposition of a closed 3-manifold. *Illinois J. Math.* **39** (1995), 358–406.

[14] LEACH, A. R. *Molecular Modeling, Principles and Applications*. Pearson Education Limited, Harlow, England, 1996.

[15] MUNKRES, J. R. *Elements of Algebraic Topology*. Addison-Wesley, Redwood City, California, 1984.

[16] RCSB. Protein data bank. http://www.rcsb.org/pdb/.

[17] ROLFSEN, D. *Knots and Links*. Publish or Perish, Inc., Houston, Texas, 1990.

[18] SEIFERT, H. Über das Geschlecht von Knoten. *Math. Annalen* **110** (1935), 571–592.

[19] TAYLOR, W. R. A deeply knotted protein structure and how it might fold. *Nature* **406** (2000), 916–919.

[20] VAN LEEUWEN, J. Finding lowest common ancestors in less than logarithmic time. Unpublished report.