Identification and Quantification of Important Voids and Pockets in Proteins

A Thesis

Submitted For the Degree of

Master of Science (Engineering)

in the Faculty of Engineering

by

Raghavendra G S

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

DECEMBER 2013

TO

My Parents

Acknowledgements

I would like to thank Prof.Vijay Natarajan for his immense help both in the professional and the personal front. I would also like to thank Prof.Narahari for his constant support especially towards the end. I am grateful to Prof.Raghavan Varadarajan for his support. I am also grateful to all my labmates for so many helpful discussions which led me forward.
Finally, I would like to express deep gratitude to my parents and friends for constant encouragement.

Publications based on this Thesis

1. Raghavendra Sridharamurthy, Harish Doraiswamy, Siddharth Patel, Raghavan Varadarajan and Vijay Natarajan. Extraction of Robust Voids and Pockets in Proteins. *EuroVis: Eurographics Conference on Visualization (Short Paper) 2013.*

2. Raghavendra Sridharamurthy, Harish Doraiswamy, Siddharth Patel, Raghavan Varadarajan and Vijay Natarajan. Extraction of robust voids and pockets in proteins. *Tech. Rep. IISC-CSA-TR-2013-3, Department of Computer Science and Automation, Indian Institute of Science 2013.* http://www.csa.iisc.ernet.in/TR/2013/3/.

Abstract

Many methods of analyzing both the physical and chemical behavior of proteins require information about its structure and stability. Also various other parameters such as energy function, solvation, hydrophobic/hydrophilic effects, surface area and volumes too play an important part in such analysis. The contribution of cavities to these parameters are very important. Existing methods to compute and measure cavities are limited by the inherent inaccuracies in the method of acquisition of data through x-ray crystallography and uncertainities in computation of radii of atoms. We present a topological framework that enables robust computation and visualization of these structures. Given a fixed set of atoms, voids and pockets are represented as subsets of the weighted Delaunay triangulation of atom centers. A novel notion of (ε, π)-stable voids helps identify voids that are stable even after perturbing the atom radii by a small value. An efficient method is described to compute these stable voids for a given input pair of values (ε, π). We also provide an implementation to visualize, explore (ε, π)-stable voids and also calculate various properties such as volumes, surface areas of the proteins and also of the cavities.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Protein molecules have a well packed structure, yet they contain cavities. These cavities play a key role in accomodating solvent molecules, resulting in interesting chemical behavior. A biomolecular model used to study this behavior typically considers various parameters such as energy function, solvation and hydrophobic effects, and surface areas and volumes.

A cavity refers to both voids (cavities without openings) and pockets (cavities with openings). Several methods have been proposed to locate such cavities in protein molecules. The input used for all such methods come from crystallographic measurements. These methods are sensitive to inaccuracies that are inherent in x-ray crystallography. Because of such inaccuracies in the measurements, a set of cavities that is reported to be in proximity may in reality be a single cavity. A very small cavity that may not exist is reported. The number of cavities may be quite different from the number reported. Figures 1.1 and 1.2 show examples of such cases in 2D via an illustration. Figures 1.3 and 1.4 show the problem as it occurs in a real world dataset. The inaccuracies may also arise due to some fundamental limitations such as the notion of radii of atoms, which is determined empirically.

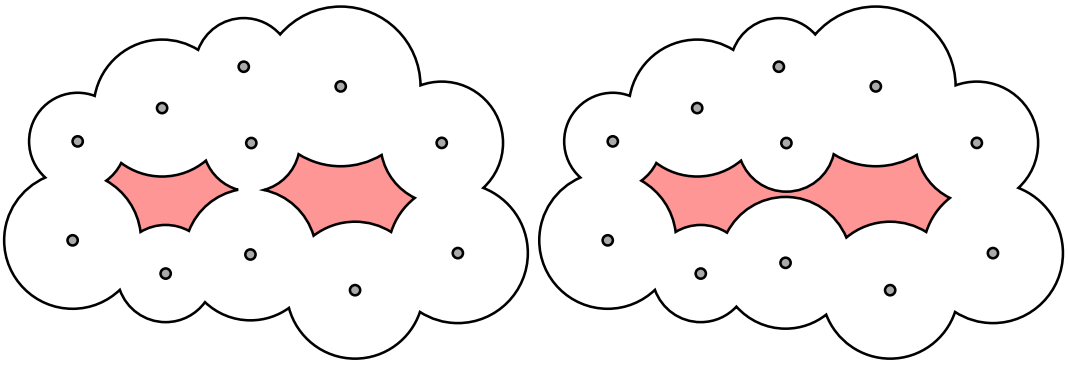


Figure 1.1: Two cavities that are apparently very near to each other may in reality be a single cavity.

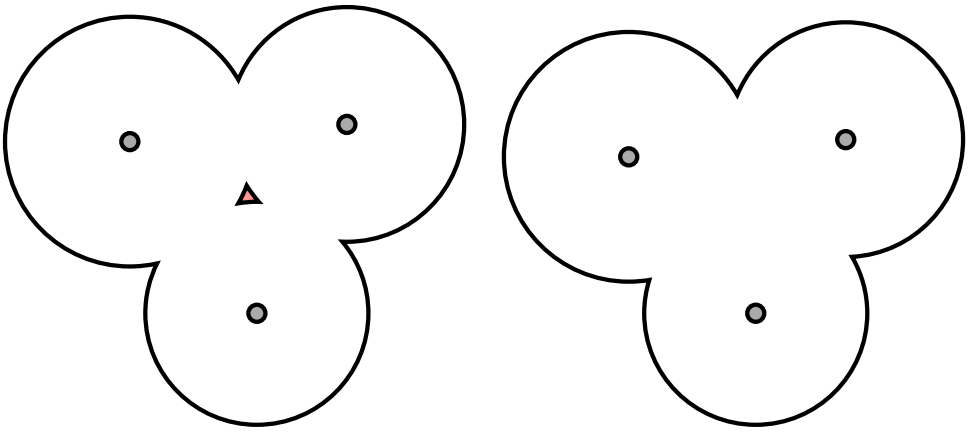


Figure 1.2: A very small cavity may be reported whereas in reality no such cavity exists.

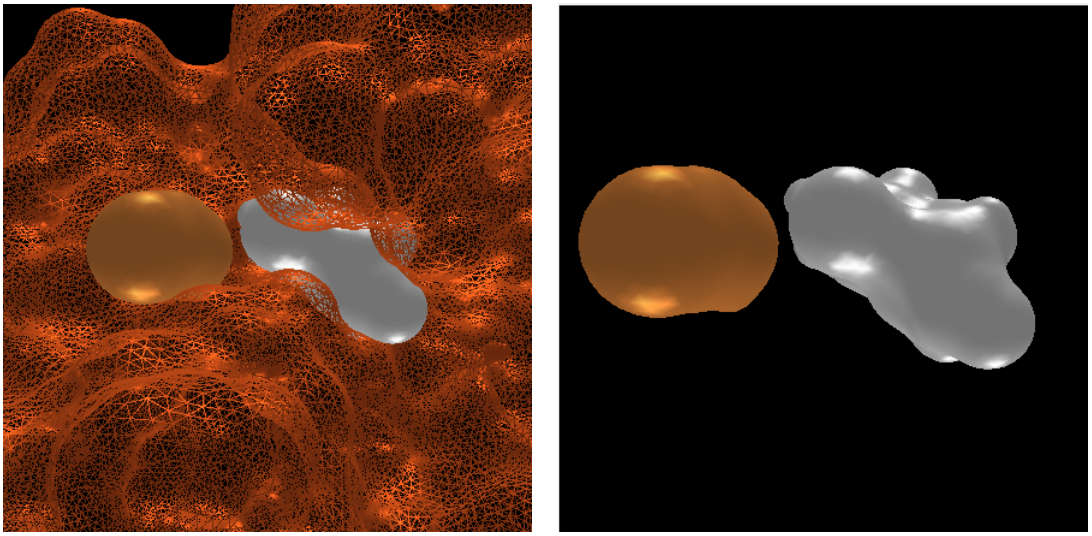


Figure 1.3: Left: Two cavities that appear very near to each other in the protein bacteriophage T4 lysozyme [PDB ID:200l]. The solid rendering shows the cavities while the wireframe represents the molecule. Right: Zoomed in view of cavities without the skin surface of the molecule.

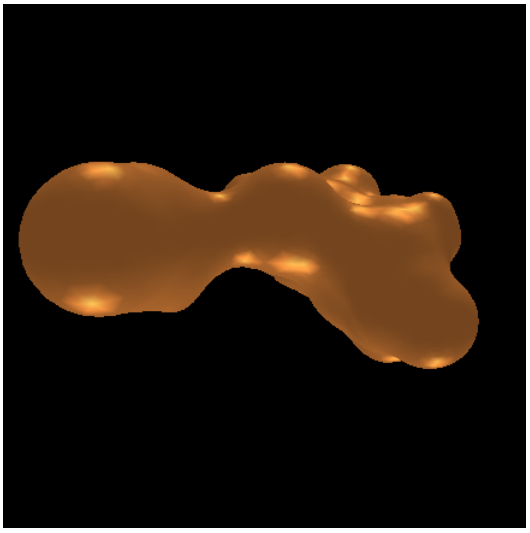


Figure 1.4: The two cavities in Figure 1.3 merged to become a single cavity

1.2 Problem

We aim to develop an interactive method to robustly compute cavities in proteins. We intend to achieve this by allowing the user to reduce, if not completely eliminate, the inaccuracies caused by the

measurement of the positions and the empirical nature of the radii mentioned in the previous section. We intend to define what robustness would mean in this context, why such a notion is important and also aim to prove the robustness of the method. Also, as an auxiliary task we aim to visualise the protein molecules and their cavities in an interactive manner and compute their properties.

1.3 Contributions

The main contributions of this thesis is in the context of robust computation of the cavities in proteins.

- We build on the existing work for cavity computation based on alpha shapes as defined by Edelsbrunner [6] and develop a method for robust computation of cavities.
- We achieve robustness by introducing a dynamic model of proteins where the radii of a select set of atoms are modified by systematically processing a filtration.
- We also show that such a modification will not violate the properties of the filtration.
- We allow, through such a modification, controlled changes in the number and properties of cavities.
- The method also supports the elimination of very small/insignificant cavities as measured by the notion of persistence defined by Edelsbrunner et al. [14].
- We develop a software to visualise the cavities and the molecule and calculate important properties such as volumes, surface areas, skin surfaces of the cavities and of the molecule.

1.4 Related work

Various approaches have been employed towards the study of biomolecular shape and structure. We focus on geometric techniques for representing the molecule and its features. In this section we describe existing geometric methods for molecular modelling and cavity computation.

Molecular models

Molecular models are important as they help us in representing the structure and attributes of molecules in a manner which is helpful for further analysis. A typical molecular model of a protein consists information about location of each atom in three dimensional space, respective Van der Waals radii, amino acids present, the organisation and sequences of those amino acids. Details can be found in Lee and Richards [22] and also in Liang et al. [23, 24].

Alpha Shape

Alpha shape representation of molecules is based on growing a set of overlapping spheres. The alpha shape describes the shape of a set of weighted points. It is a good representation for calculating

topological properties. Increasing/decreasing α corresponds to growing/shrinking the spheres. Location of the atoms and their radii are extracted from the molecular model which results in a 3D pointset and alpha shape spectrum is computed for the resulting point set. Alpha shapes were defined on both 2D and 3D pointsets both unweighted and weighted firstly by Edelsbrunner et al. [12] and then Edelsbrunner and Mücke [11], Edelsbrunner [6]. Betti numbers give us the cardinality of interesting topological features. Definition and methods to compute betti numbers efficiently with respect to alpha shapes were defined by Delfinado and Edelsbrunner [4].

Persistence

The variation of the parameter α results in changes in topological features. Features may be created/destroyed. Persistence which can roughly explained as the "lifetime" of specific feature of interest in topological analysis was first defined with respect to alpha shapes by Edelsbrunner et al. [14]. Persistence gives us a useful way to extract and classify features present in datasets. Persistence is applied as an additional refinement criteria to the set of cavities.

Properties of Molecules

Properties of molecules such as volumes, surface areas, volumes and surface areas of the cavities are very important for the problem defined. We calculate these properties to provide a better understanding of the interesting cavities and also the protein molecule. Several methods have been proposed to compute these metrics including those by Liang et al. [23, 24], Dundas et al. [5], Edelsbrunner [7], Pavani and Ranghino [29], Gibson and Scheraga [15], Edelsbrunner and Fu [8].

Skin Surface

Skin surface of a molecule provides a very good model for visualisation of the molecules. Skin surface has some very nice properties in terms continuity which helps in achieving an informative visualisation. Properties and computation of Skin Surfaces can be understood thoroughly by studying the work of Cheng and Shi [3]. We use the implementation provided along with Cheng and Shi [3] to compute skin surface of molecules and the cavities.

Cavity computation

Cavity computation has been a very active area of research and hence numerous methods have been proposed till now. We focus our attention on geometric methods. Edelsbrunner et al. [8, 9] and Liang et al. [24, 25] propose a definition that is based on the theory of alpha shapes and discrete flows in Delaunay triangulations. Kim et al. [17, 16] propose a definition of cavities based on an alternate representation of a set of atoms called beta shapes that faithfully captures proximity. Tools based on the above approach are available and widely used [5, 19, 18]. Till and Ullmann [34] employ a graph theoretic algorithm to identify cavities and compute their volume. Varadarajan et al. [2] employ a Monte Carlo procedure to position water molecules together with a Voronoi region-based method to locate empty space. They discuss the importance of accurate identification of cavities for the study of protein structure

and stability. Novel Voronoi diagram-based techniques for the extraction and visualization of cavities have also been developed from the viewpoint of studying and interactively exploring access paths to active sites [31, 30, 26, 27]. Krone et al. [20] present a visualization tool for interactive exploration of protein cavities in dynamic data and in [21] uses gaussian density surfaces to extend their previous work. While gaussian density surface is C^1-Continuous and thus fares better than C^0-Continuous counterparts such as molecular surface(MS), do not represent the individual atoms. So, they require recomputation if there is a mutation where a residue is replaced by another. Also, the volume of the molecule and the cavities are not explicitly stored. Zhang and Bajaj [35] presents a technique which uses two-step level set propagation method to extract pockets from a closed compact smooth surface representation of protein molecules. While the algorithm works for any compact smooth surface representation of molecules the surfaces in turn are constructed using empirical radii such as vander Waals radii and thus also has same drawback which is present in the other methods.

Several softwares are available for detection and quantification of voids. The original implementation based on alpha shape theory, *alvis* by Edelsbrunner and Mücke [11] and its variants *proshape* by Koehl et al. [19] and *CASTp* by Dundas et al. [5] are widely used.

1.5 Organisation of the Thesis

The rest of the thesis is organised as follows. Chapter 2 defines all the important concepts and provides the relevant background. Chapter 3 describes an algorithm for modifying the filtration built upon alpha shapes. Chapter 4 discusses the implementation. Chapter 5 discusses the results. Chapter 6 concludes the thesis.

Chapter 2

Background

We rely on geometric structures for representing the molecules and computating robust voids. Prior to the description of the method for computing robust voids, we introduce the geometric structures that have been developed to describe the shape of the biomolecules. Each section describes a structure together with its properties.

2.1 Molecular models

Three definitions of surfaces are widely used for representing biomolecules as described in Lee and Richards [22]. The *van der Waals surface (VS)* is the envelope of atoms that are represented by spherical balls with van der Waals radii (Figure 2.1 (a)). The *solvent accesible surface (SAS)* is generated by the centre of the solvent (modeled as a rigid sphere) when rolling over the van der Waals surface of the molecule (Figure 2.1 (b)). The *molecular surface (MS)* is generated by the front of the solvent sphere as it rolls over the van der Waals surface (Figure 2.1 (c)).

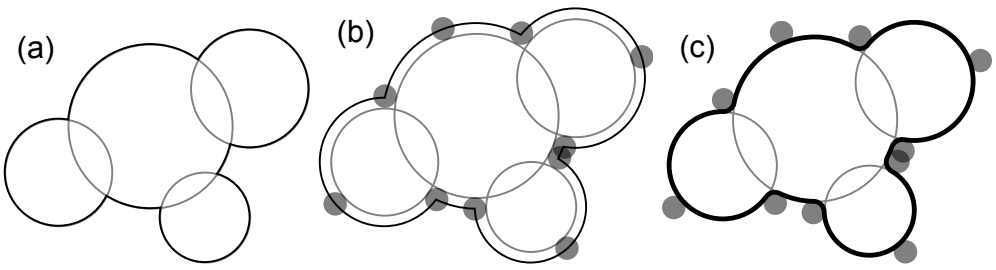


Figure 2.1: Molecular Surface models **(a)**. Van der Waals Surface (VS) **(b)**. Solvent Accesible Surface (SAS) **(c)**. Molecular Surface (MS)

2.2 Simplices and simplicial complexes

Simplices and *simplicial complexes* as defined in Munkres [28] are used to represent a topological space.

Simplex: A *k-simplex* σ is the *convex hull* of $k+1$ affinely independent points, *conv* $\{u_0, u_1, \ldots, u_k\}$ with *dimension* dim $\sigma = k$.

We use special names for the first few dimensions, *vertex* for 0-simplex, *edge* for 1-simplex, *triangle* for 2-simplex, and *tetrahedron* for 3-simplex, see Figure 2.2. A *face* of σ is the convex hull of a non-empty subset of the set of $k+1$ points.A face is proper if the subset is proper. We denote $\tau \leq \sigma$ if τ is a *face* and $\tau < \sigma$ if it is proper.

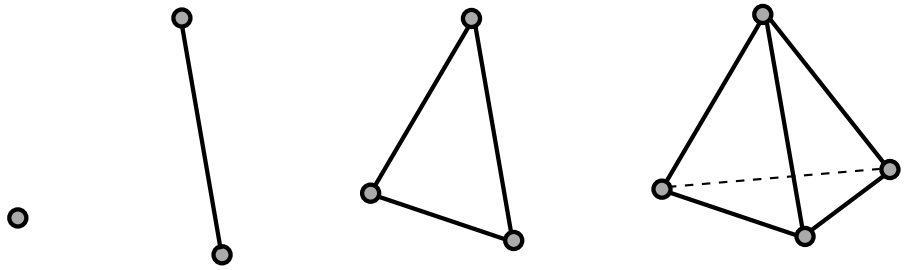


Figure 2.2: k-simplices for k = 0,1,2,3

Simplicial Complex: A *simplicial complex* is a finite collection of simplices K such that $\sigma \in K$ and $\tau \leq \sigma$ implies $\tau \in K$, and $\sigma_1, \sigma_2 \in K$ implies $\sigma_1 \cap \sigma_2$ is either empty or a face of both.

The *dimension* of K is the maximum dimension of its simplices and the *underlying space* denoted as $|K|$ is union of its simplices together with the topology inherited by $\mathbb{R}^d$. A *subcomplex* of K is a simplicial complex $L \subseteq K$. A particular subcomplex of K is the *j-skeleton* consisting of all simplices of dimension j or less, $K^{(j)} = \{\sigma \in K \mid dim\, \sigma \leq j\}$.

2.3 Voronoi diagrams and Delaunay triangulations/complexes

Delaunay complexes are specific simplicial complexes that have some useful properties and hence used in most of the geometric models. *Voronoi diagrams* are the dual of delaunay complexes. For details see Edelsbrunner et al. [12].

Let $S \subseteq \mathbb{R}^d$ be a finite set of points.

Voronoi Cell: The *voronoi cell* of a point $p \in S$ is the set of points in $\mathbb{R}^d$ that are closer to p than to other points in S. i.e. $V_p = \{x \in \mathbb{R}^d \mid \|x - p\| \leq \|x - q\|, \forall\, q \in S\}$.

The Voronoi cell of p is the intersection of half-spaces of points that are at least as close to p as to q for all points $q \in S$. So V_p is a convex polyhedron in $\mathbb{R}^d$. Any two voronoi cells share at most a common

subset of their boundary and together the voronoi cells cover the entire space as shown in Figure 2.3 (b). The resulting diagram is known as the *voronoi diagram*.

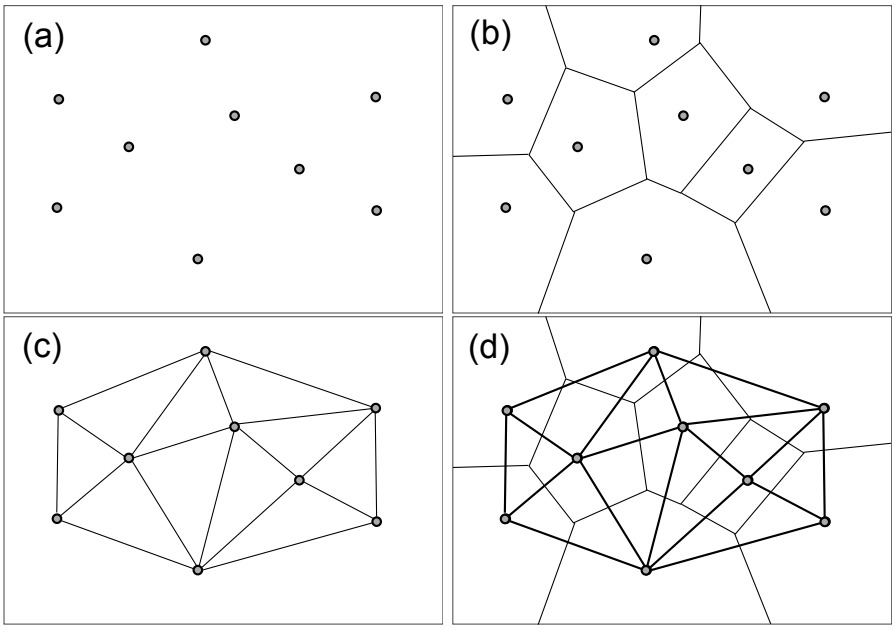


Figure 2.3: **(a)**. Point Set. **(b)**. Voronoi Diagram. **(c)**. Delaunay Triangulation. **(d)**. Voronoi and Delaunay complex overlayed.

Consider the generalisation of the voronoi diagram where the points have weights. Let w_p be the weight of a point $p \in S$, the *weighted square distance* or *power distance* of a point $x \in \mathbb{R}^d$ from p is given by $\pi_p(x) = \|x - p\|^2 - w_p$. The *bisector* of two weighted points is the set of points with equal power distance from both as shown in Figure 2.4. *Weighted voronoi cell* or *power cell* is defined as set points in $\mathbb{R}^d$ for which p is the closest weighted point. i.e. $V_p = \{x \in \mathbb{R}^d \mid \pi_p(x) \leq \pi_q(x),\ q \in S\}$. Add edges in such a way that there is an edge between two points u and v iff their voronoi cells share boundaries. Then assuming general position, we get a geometric realisation or triangulation as shown in Figure 2.3 (c).

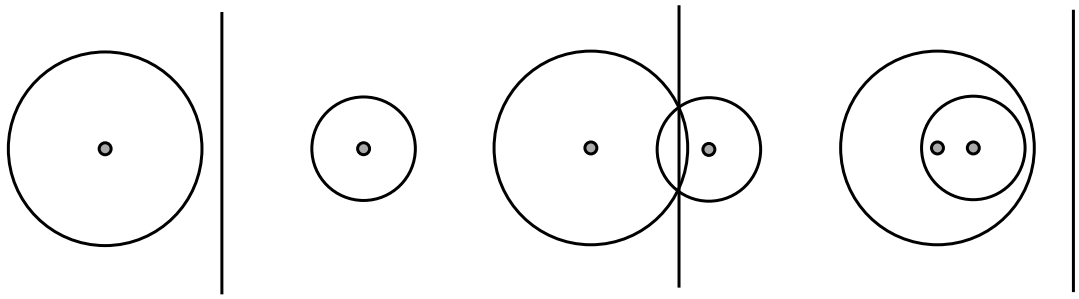


Figure 2.4: The bisector of two weighted points is the collection of points that are equidistant, in terms of power distance from both the points.

Delaunay triangulation: Formally *delaunay complex* or *triangulation* for $S \subseteq \mathbb{R}^d$ is defined as the *nerve* or *dual* of voronoi diagram,
$D = \left[\sigma \in S \mid \bigcap_{p \in \sigma} V_p \neq \emptyset\right]$.

The weighted case can similarly be constructed and is called the *weighted delaunay triangulation* or *regular triangulation*.

2.4 Alpha complexes

Molecules are often represented using a space-filling model such as a union of balls. The weighted Voronoi diagram may be extended to represent the contribution from each atom to the union of balls. Consider an atom p. Define B_p as an open ball having the radius of the atom p. Let V_p be the weighted Voronoi cell corresponding to p, where the weight is equal to the square of the atom radius. The contribution from each atom p is equal to $B_p \cap V_p$, the intersection between the ball corresponding to the atom and the weighted Voronoi cell of p. The corresponding dual structure is a subcomplex of the weighted Delaunay triangulation and called the *dual complex*, see Figure 2.5.

Edelsbrunner et al. [12, 11, 6] consider a growth model where the ball radii grow, and track the changes in the dual complex. The growth parameter, α, corresponds to a radius $\sqrt{r_p^2 + \alpha^2}$ for a ball centered at p with radius r_p. The weight of the point $w(p)$ increases to $w(p) + \alpha^2$. Note that $\alpha = 0$ corresponds to no growth. The dual complex corresponding to a set of balls after they are grown by α is called the *alpha complex*.

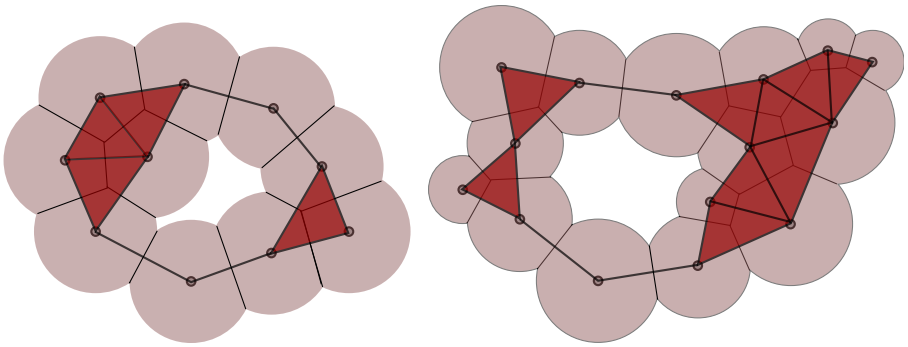


Figure 2.5: Union of disks intersected by voronoi regions and the dual alpha complexes. (Left) unweighted (Right) weighted

Given a simplex $\sigma = \{p_0, p_1, ..., p_k\}$ and any $p \in \sigma$ is represented as a weighted point $p = (p_c, p_r)$, let $y = (y_c, y_r)$ be the point with minimum weight orthogonal to all $p \in \sigma$. y is called the *orthogonal center* of σ. Also y_r is called the *size* of the simplex σ.

Filtration: Writing K^i for the i^{th} alpha complex in sequence we get $\emptyset = K^0 \subset K^1 \subset K^2 \ldots \subset K^m = D$ which is called a *filtration*.

We can also write this filtration in terms of α. Assume each K^i corresponds to a particular value α_i. Then we have $\emptyset = K(-\infty) = K(\alpha_0) = K^0 \subset K^1 ... \subset K^m = K(\alpha_m) = K(+\infty) = D$

Rank: The rank of a simplicial complex refers to its position in the filtration.

In case of the alpha complex, the difference between K^{i+1} and K^i is either a simplex or a set of simplices. A filtration can be constructed by adding exactly one simplex at a time, in which case the simplices between consecutive alpha complexes are sorted according to their dimensions and added one at a time.

2.5 Homology and betti numbers

We use *homology groups* and *betti numbers* as described in Delfinado and Edelsbrunner [4] to mathematically quantify intuitive notions like tunnels and voids.

Let K be a simplicial complex. We construct groups by adding the simplices in K. A set of k-simplices is known as a *k-chain*. The sum of two *k-chains* is defined as the *symmetric difference* between two sets. C_k is the set of *k-chains* and $(C_k, +)$ is group of *k-chains* under addition. The identity of this group is the empty set. We connect chain groups of different dimensions by homomorphisms that maps chains to their boundaries. The *boundary* of a k-simplex σ is given by $\partial\sigma = \{\tau \leq \sigma \mid dim\, \tau = dim\, \sigma - 1\}$. The *boundary* of a chain is equal to the sum of the boundaries of its simplices $\partial c = \sum_{\sigma \in c} \partial\sigma$. We thus have

boundary homomorphism $\partial = \partial_k : C_k \longrightarrow C_{k-1}$. *Chain complex* of K is given by sequence of *chain groups* connected by *boundary homomorphism.* $\ldots \longrightarrow^{\partial_{k+2}} C_{k+1} \longrightarrow^{\partial_{k+1}} C_k \longrightarrow^{\partial_k} C_{k-1} \longrightarrow^{\partial_{k-1}} \ldots$

A *k-cycle* is a *k-chain* c with boundary 0. i.e. $\partial c = 0$. The set of *k-cycles* is the *kernel* of the *k-th boundary homomorphism* i.e. $Z_k = ker\ \partial_k$. Two *k-cycles* add up to another *k-cycle* implying $(Z_k, +)$ is a subgroup of $(C_k, +)$. A *k-boundary* is a *k-chain* c for which there exist a *(k+1)-chain* d with $\partial d = c$. The set of *k-boundaries* is the *image* of *(k+1)-st homomorphism* i.e. $B_k = im\ \partial_{k+1}$. Two *k-boundaries* add up to another *k-boundary* implying $(B_k, +)$ is a subgroup of $(C_k, +)$. The boundary of a boundary is always empty (refer Delfinado and Edelsbrunner [4]) i.e. $\partial\partial d = 0$ which implies that $(B_k, +)$ is subgroup of $(Z_k, +)$. See Figure 2.6.

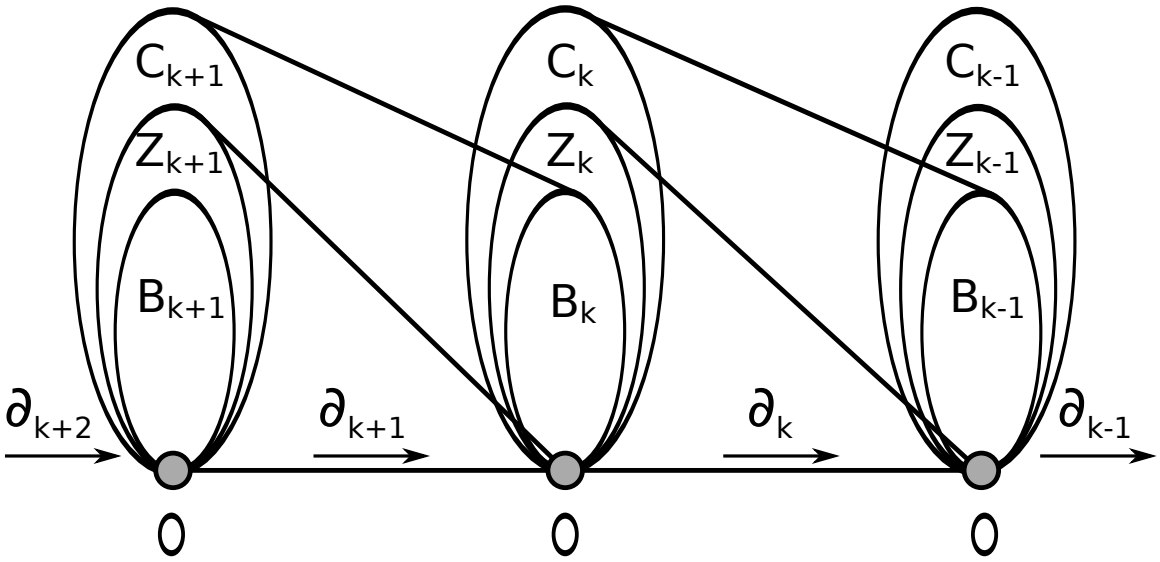


Figure 2.6: Chain complex and groups of cycles and boundaries

Homology groups: The *kth homology group* is given by $H_k = Z_k/B_k$.

The rank of this group is known as the *kth betti number* $\beta_k = rank\ H_k = rank\ Z_k - rank\ B_k$. Intuitively β_0 is the number of components, β_1 is the number of tunnels and β_2 is the number of voids. Calculation of Betti numbers involve matrix reductions and hence can be costly but for the special case of complexes in $\mathbb{R}^3$ an *incremental approach* efficiently computes the betti numbers (details in Delfinado and Edelsbrunner [4]).

2.6 Flow relation

Edelsbrunner et al. [13] defines delaunay flow and depth and uses them to compute cavities.

Let T' be a set of tetrahedra in the delaunay triangulation(D) and $T = T' \cup \{\tau_\infty\}$ where τ_∞ denotes

a dummy tetrahedron representing the complementary space.

Flow: A flow relation $' \prec ' \subseteq T \times T$ is defined between a pair of tetrahedra τ, σ. $\tau \prec \sigma$ if τ and σ share a common triangle φ and interior of τ ($int\,\tau$) and orthogonal center (as defined in Section 2.4) z_τ of τ lies on different sides of the plane affine to φ. The flow relation $\prec$ is acyclic and its transitive closure is transitive

If $\tau \prec \sigma$ we call τ as *predecessor* of σ and σ as *successor* of τ. The *descendent set* of τ is $Des\,\tau = \{\tau\} \cup \bigcup_{\tau \prec \sigma} Des\,\sigma$. The *ancestor set* of σ is $Anc\,\sigma = \{\sigma\} \cup \bigcup_{\sigma \succ \tau} Anc\,\tau$.

Depth: If σ_j is a tetrahedron where j denotes its position in the filtration then *depth* is defined as $dp\,\sigma_j = max\,\{k \mid \sigma_k \in Des\,\sigma_j\} = max(\{j\} \bigcup \{dp\,\tau \mid \sigma_j \prec \tau\})$.

2.7 Voids and pockets

Voids: *Voids* are defined as components of the complement of the union of balls which is a part of three dimensional space not covered by any of the balls except the infinitely large outside component. see Figure 2.7 (a). Voids in K^i are represented by simplices of delaunay complex which are not present in K^i. The algorithms to find voids and calculate volumes and surface areas are discussed in Edelsbrunner and Fu [8], Liang et al. [23, 24] and Chakravarty et al. [2].

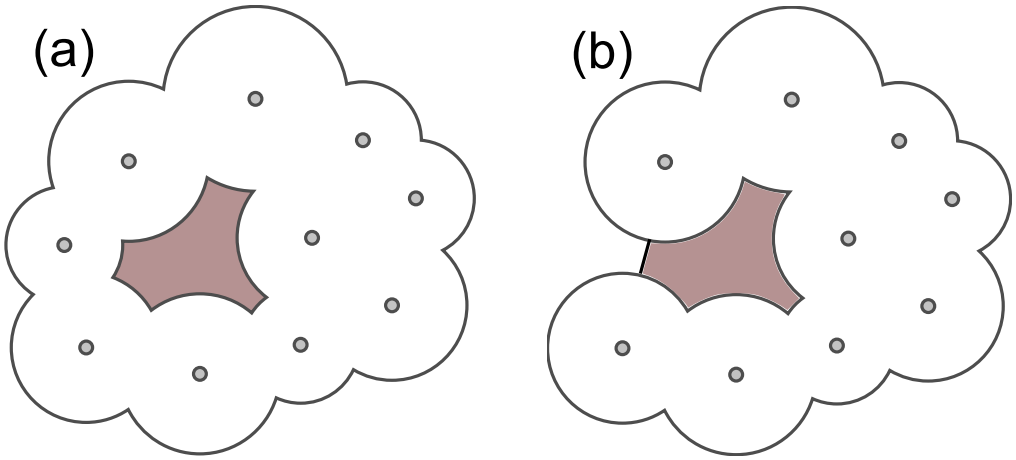


Figure 2.7: **(a)** Voids and **(b)** Pockets computed as the connected component of the complement of union of balls.

Pockets: *Pockets* are generalisations of voids, obtained by relaxing the constraint that they should be disconnected from the infinite outer component. see Figure 2.7 (b). However not all connected components of the complement are pockets. A key requirement is that a pocket is wider in the inside than at possible entrances from the outside so that the narrow entrance closes before the interior disappears (for details see Liang et al. [25]). In other words a pocket turns into a void before it disappears if we

grow the balls. Detection and quantification of pockets using discrete flow methods is described by Edelsbrunner et al. [13].

2.8 Persistence

Persistence is used to measure the lifetime of features such as voids and pockets. High persistent voids/pockets are naturally of interest.

Assume a filtration (2.4) where simplices are added one at a time i.e $\emptyset = K^0 \subset K^1 \subset \ldots \subset K^m = K$ with $K^p = K^{p-1} \bigcup \sigma^p$.

k-th p-persistent homology group: The *k-th p-persistent homology group* of K^i is defined as $H_k^{i,p} = Z_k^i / \left(B_k^{i+p} \bigcap Z_k^i\right)$. *Persistence* measures the lifetime a cycle in Z_k^i i.e. the time between its creation and when it becomes a boundary when simplices are inserted one at a time in the filtration order. If it doesn't become a boundary till K^{i+p} then it "persists" till that level. Let z be a non-bounding cycle created by a simplex σ^i when it enters at time i. The cycle z "creates" a *class* of *homologous cycles* $[z]$. A simplex σ^j "destroys" z and $[z]$ if it transforms it into a boundary.

Persistence: *Persistence* is the difference between the stage where a cycle is created and when it is destroyed. The persistence of z and its homology class $[z]$ is given by $j - i$. σ^i is the *creator (positive simplex)* and σ^j is the *destroyer (negative simplex)* of $[z]$. If a cycle doesn't have a destroyer then its persistence is ∞. Given a filtration the persistence of cycles can be computed efficiently using a union-find data structure.

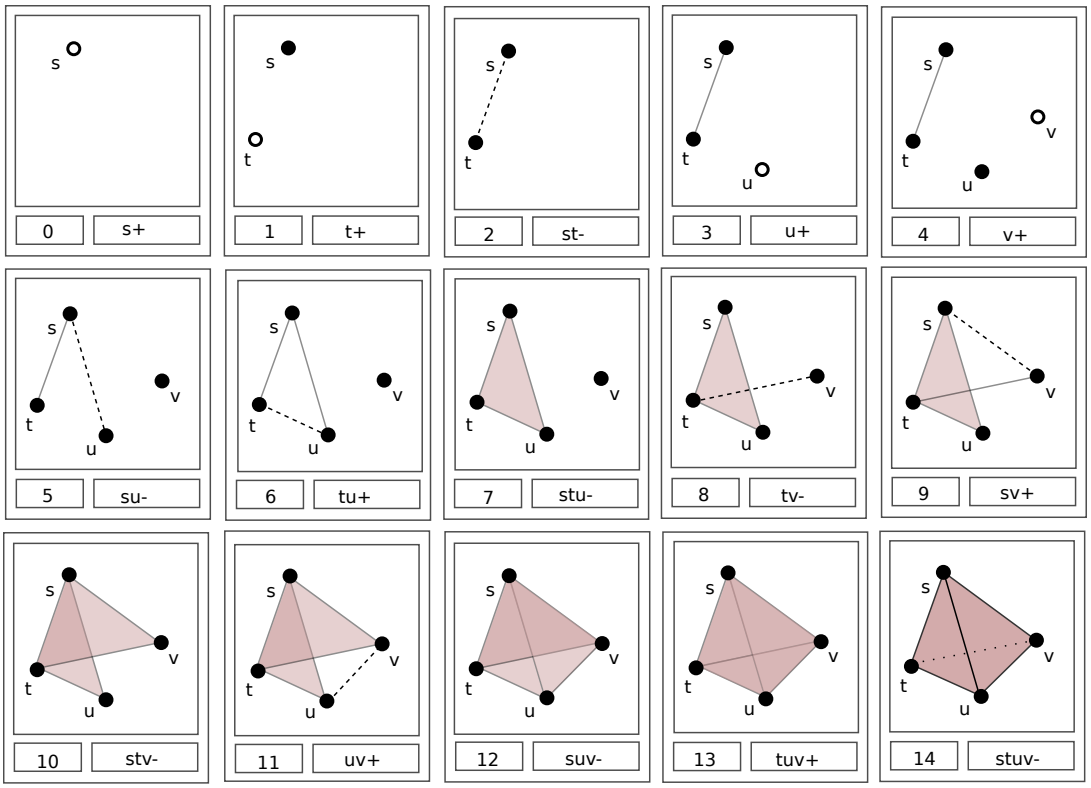


Figure 2.8: An example for filtration illustrating addition of simplices in a particular order. Also the arrival time and the behavior of the simplices are shown in the boxes

To illustrate how the filtration affects persistence we compute persistence of the cycles in the filtration shown in Figure 2.8 as follows. Firstly, in each box the number denotes the timestamp and the simplex which is added is denoted in the adjacent box along with a positive or a negative sign which respectively signifies an addition or a deletion of a cycle.

The $0-cycle$ or the component s enters the filtration at the time instance 0 and it gets destroyed at the time instance 2 when the edge st appears. So the persistence of s is $2-0=2$. The $0-cycle$ v appears at 4 and gets destroyed at the instance 8 when the edge tv appears. So the persistence of v is $8-4=4$. The $1-cycle$ stu enters the filtration when the edge tu appears at 6 and gets destroyed at 7 when the triangle stu appears. So persistence of $cycle$ stu is $7-6=1$. The $2-cycle$ $stuv$ enters the filtration when the triangle tuv appears at 13 and gets destroyed at 14 when the tetrahedron $stuv$ appears and hence its persistence is $14-13=1$. Note that if we had interchanged the events occuring at the timestamps 7 and 8 (i.e. appearance of edge tv and triangle stu) it wouldn't violate the filtration property but the persistence of $0-cycle$ v would have decreased to 3 and persistence of $1-cycle$ stu would have increased to 2.

2.9 Skin surface

Skin surfaces are used to visualise molecules because of their particularly nice properties that enhance the perception of the surface. For example the skin surface is C^1-continuous and its maximum curvature is also continuous.

Skin surface: A *skin surface* F_B is defined by a set of spheres B in $\mathbb{R}^3$. It is the envelope of an infinite family of spheres derived from B via convex combinations and shrinking. Even though the family of spheres is infinite, the skin surface can be decomposed to a collection of quadratic patches.

The skin model of a molecule is the skin surface specified by the set of spheres positioned at each atom coordinate and equipped with a radius equal to $\sqrt{2}$ times the summation of the atom's van der Waals radius and the radius of the probe sphere. The probe sphere radius is typically chosen as 1.4Å to represent water.

Figure 2.9 shows an example of rendering of skin surface for a protein molecule.

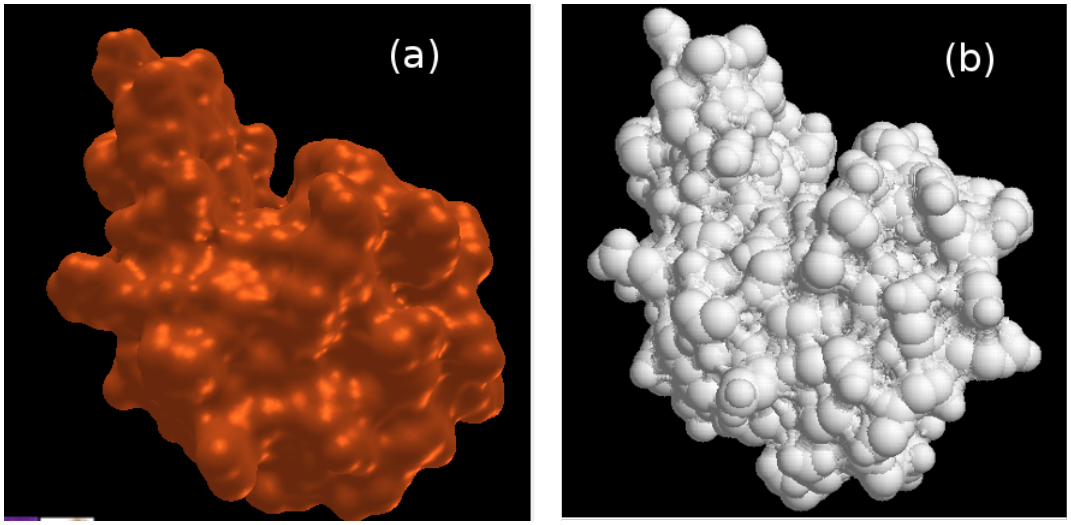


Figure 2.9: Comparison between the skin surface and the molecular surface (rendered by rasmol) of the protein human lysozyme [PDB ID:1OUB]

Chapter 3

Robust voids and their Computation

In this chapter we introduce a notion of robust/stable voids and a method to achieve this robustness or stabiltiy by modifying the filtration. We explain the intuition behind the method, describe the method in detail, discuss few refinements, and also prove the correctness of the method.

3.1 Stablity/robustness of voids

We introduce a notion of robustness based on two parameters, one local and another global. The local parameter is referred to as stability and the global parameter is specified by topological persistence. In order to simplify the description, we assume that the α-value of interest is $\alpha = 0$. However, the definitions, methods, and subsequent analysis are valid for all values of α.

3.1.1 ε-stable voids

Consider the interval $[-\varepsilon, \varepsilon]$ of α values, where $\varepsilon \geq 0$. A void is called an *ε-stable* void if it remains a single connected void within all α-complexes for α values in the range $[-\varepsilon, \varepsilon]$. In other words, using the lifetime terminology, the cavity is born, possibly split into multiple components, and destroyed at α-values that lie strictly outside of this interval. This set is the target we intend to achieve from our modification. The voids here are precisely the ones we have discussed in Section 1.1 illustrated by Figure 1.1.

3.1.2 π-persistent voids.

A void is called a *π-persistent* void if the topological persistence of its creator (a 2-cycle) is greater than π. In other words, the size of the void measured in terms of its lifetime is greater than π.

3.1.3 (ε, π)-stable voids

(ε,π)-stable voids: Combining the two notions of robustness, we call a void to be (ε, π)-*stable* if it is both ε-stable and π-persistent. Here the intuition is to restrict the ε-stable voids further to eliminate voids with low topological persistence. This captures the case discussed in Section 1.1 and illustrated by Fig 1.2. The above definitions help measure the stability of the voids when the radii are perturbed by a small value. The local parameter considers perturbation within a small interval centered at the α-value of interest whereas the global parameter essentially measures the size of the void in terms of its lifetime in the filtration. Voids of interest may often not be stable with respect to both notions. For example, a large sized void (π-persistent for some large value of π) may be born within the interval $[-\varepsilon, \varepsilon]$. However, note that a small perturbation in the radii of atoms that line the surface of the void could result in an earlier birth time, hence making the cavity to be ε-stable. We aim to extract all voids that are either stable as is or can be made stable via a small perturbation. The former can be identified directly from the persistence algorithm [14], which labels creators and destroyers of the voids (2-cycles). We next describe a structured approach to compute the latter set of stable voids.

3.2 Modifying the filtration

In this section we explain the need to modify the filtration, how we modify the filtration, and how such modifications affect the voids and their properties.

3.2.1 The need for the modification

The filtration of the weighted Delaunay triangulation as defined by the α-values provides an explicit representation of the birth/death times of each void and the evolution during its lifetime. We propose to alter the birth/death times of voids by modifying the filtration instead of directly modifying radii of atoms that line the surface of the void. While the latter approach follows directly from the definition, it is cumbersome and computationally inefficient. For example, varying the radii without explicit control may lead to changes in the triangulation and the alpha complex. These changes need to be explicitly tracked, else they may lead to inconsistencies between the alpha complex that represents the molecule and the space-fill model. Resolving these inconsistencies would necessitate the recomputation of all representations. On the other hand, the former approach is simpler and computationally efficient as shown below. Before discussing the modification, we discuss how changes in the α value results in changes in the rank and how such changes affect the alpha shape.

3.2.2 Tracking rank changes

One or more simplices are inserted to obtain a rank $i+1$ simplicial complex from a rank i simplicial complex in the filtration. Higher ranks correspond to higher values of α. The topology of voids may change when the simplices are inserted. These topology changes may therefore be avoided by delaying the insertion of the simplices. We first discuss the possible topology changes caused by the insertion of a k-simplex.

Case $k = 0, 1$**.** Insertion of a vertex or an edge does not affect the voids.

Case $k = 2$**.** A triangle may either not affect any void, split a void into two or create a new void. A triangle lies on the boundary of at most two voids and hence its insertion may have caused the split of one void into at most two voids. Figure 3.1(a) illustrates the split event in 2D. Inserting a triangle (thick solid edge in the 2D illustration) splits the void into two.

Case $k = 3$**.** A tetrahedron always destroy a void. All triangles bounding the tetrahedron are inserted prior to the tetrahedron and constitute the surface of the void.

We make a few observations here. In the case $k = 2$ a triangle is at most common to two voids, so its introduction can only split a void into two voids. Note that a single triangle introduced into the alpha complex can never split a void into more than two voids. Also, in case $k = 3$, there should already exist a void that is filled by the tetrahedron i.e.its four bounding triangles should already be present in the alpha complex and hence a tetrahedron always destroys a void.

3.2.3 Delayed simplex insertion

We detect simplices that cause topology changes and optionally delay their insertion with the aim of computing stable voids. Simplicial complexes in the filtration of the weighted Delaunay triangulation and the order of simplices that are inserted to generate the filtration satisfy several containment and incidence properties. These properties are to be satisfied for the modified filtration as well.

We are interested particularly in the interval $[-\varepsilon, \varepsilon]$. Let K^j be the alpha complex for $\alpha = -\varepsilon$ and K^l be the alpha complex for $\alpha = \varepsilon$. Now for $\varepsilon > 0$ we have $\varepsilon > -\varepsilon$ and hence $K^j \subset K^l$. We consider the filtration readily provided by the alpha complex and modify it. We identify the set of triangles (Σ_t) and tetrahedra (Σ_T) that are inserted for values of α in the range $[-\varepsilon, \varepsilon]$. From the sets Σ_t we extract the set of triangles $\Sigma_t^{'}$, where each triangle $\sigma \in \Sigma_t^{'}$ splits a void. We add the tetrahedra that form the cofaces of triangles belonging to $\Sigma_t^{'}$ to Σ_T to construct a new set of tetrahedra $\Sigma_T^{'}$. We delay the insertion of simplices σ_i in $\Sigma_t^{'}$ and $\Sigma_T^{'}$ such that $\sigma_i \notin K^j$ but $\sigma_i \in K^l$, where $K^j \subset K^l \subset D$. This seemingly arbitrary delayed insertion might violate the filtration property. We can avoid the violation in two different ways as described below.

3.2.3.1 Earliest Coface Search Method

1. Move all tetrahedra in $\Sigma_T^{'}$ to the end of the filtration. All such tetrahedra are contained in D but not in $K^m \subset D$ for all m.
2. For each triangle in $\Sigma_t^{'}$, find its coface tetrahedra τ_1 and τ_2, where τ_1 appears before τ_2 in the original filtration.
3. Delay the insertion of the triangle until the rank at which the tetrahedron τ_1 is inserted.

3.2.3.2 Conservative Method

1. Move all tetrahedra in $\Sigma_T^{'}$ to the end of the filtration. All such tetrahedra are contained in D but not in $K^m \subset D$ for all m.
2. For each triangle in $\Sigma_t^{'}$, find its coface tetrahedra τ_1 and τ_2, where τ_1 appears before τ_2 in the original filtration.
3. Delay the insertion of the triangle and the two tetrahedra, τ_1 and τ_2 to the end of the filtration.

Note that in both the approaches the first two steps are the same and the difference lies only in Step 3. The approach described in 3.2.3.2 is conservative in the sense that we move all the simplices to the end of the filtration which is more than what is required. We follow 3.2.3.2 rather than 3.2.3.1 to save time as described in Section 4.10.

3.3 Detailed explanation

Consider the delayed insertion of a triangle that causes a void to split. The void is no longer split into two and instead reported as a single connected ε-stable void. The delay corresponds to shrinking the atoms centered at the vertices of the triangle. However, note that the radii are not yet modified. We optionally modify the radii later for further analysis of the void. A triangle that creates a void is left untouched and the corresponding void is also declared to be ε-stable. The triangle insertion may be advanced to ensure that the void is created outside the interval. This corresponds to a small increase in the radii of the atoms centered at the vertices of the triangle. We choose not to explicitly advance the triangle insertion because it does not affect the results for small values of ε.

To further illustrate the modification in 2D consider the Figure 3.1(a).

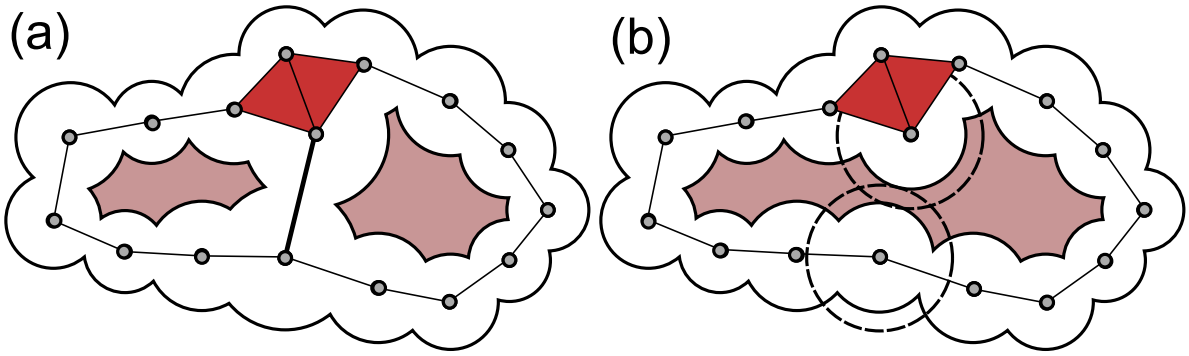


Figure 3.1: 2D illustration of simplex insertion causing a void to split. **(a)**. Voids occur near to each other and the edge that splits the single void into two. **(b)**. The filtration is modified by delaying the insertion of the edge. After modification, the two voids merge into one.

Assume that the edge (triangle in 3D) highlighted becomes part of the alpha complex in the interval $[-\varepsilon, \varepsilon]$. Further it also satisfies the criterion that it bounds two different tunnels (voids in 3D). So it becomes a candidate for the modification discussed in section 3.2.3. Now, we move the edge to the end of the filtration which means that it will no more be a part of the alpha complex as shown in Figure 3.1(b).

Also the radii of the atoms in the corners of the edge (triangle in 3D) is changed accordingly in relation to the ε value. So, in this way, selective modification of the radii of a specific set of atoms is achieved in a controlled manner.

3.4 Algorithms

We discuss the algorithms for filtration modification along with a proof of correctness and analysis of runtime complexity.

3.4.1 Generating the set of candidate simplices

Algorithm 3.1 Generate candidate simplices set

CANDIDATE-SET(ε)

1. Find $r_1 = rank(-\varepsilon)$ and $r_2 = rank(\varepsilon)$
2. Initialise $\Sigma_t = \emptyset$ and $\Sigma_T = \emptyset$
3. for ranks r varying from r_1 to r_2
   (a) if a simplex $\sigma_i \in K^r \setminus K^{r-1}$
      i. if $dim(\sigma_i) = 2$ add σ_i to the set Σ_t
      ii. else if $dim(\sigma_i) = 3$ add σ_i to the set Σ_T

3.4.2 Refining the set of candidate simplices

Algorithm 3.2 Refining the set of Candidate simplices

REFINE-CANDIDATE-SETS(Σ_t, Σ_T)

1. Initialise $\Sigma_t^{'} = \emptyset$ and $\Sigma_T^{'} = \Sigma_T$
2. For each $\sigma_i \in \Sigma_t$
   (a) find cofaces τ_1, τ_2
   (b) find cavities p_1 and p_2 such that $\tau_1 \in p_1$ and $\tau_2 \in p_2$
   (c) if $p_1 \neq p_2$
      i. add σ_i to the set $\Sigma_t^{'}$
      ii. add τ_1, τ_2 to the set $\Sigma_T^{'}$

3.4.3 Modifying the filtration

Algorithm 3.3 Filtration modification

MODIFY-FILTRATION($\Sigma_t^{'}, \Sigma_T^{'}, F_{\mathcal{A}}$)

1. For each $\sigma_i \in \Sigma_t^{'}$
   (a) find position of σ_i in alpha complex filtration $F_{\mathcal{A}}$.
   (b) move σ_i to the end of $F_{\mathcal{A}}$
2. For each $\sigma_i \in \Sigma_T^{'}$
   (a) find position of σ_i in alpha complex filtration $F_{\mathcal{A}}$.
   (b) move σ_i to the end of $F_{\mathcal{A}}$

3.4.4 Recomputing voids

After the filtration modification we recompute the voids. A brief discussion of the method together with the implementation details appears in Section 4.6. For full description of the algorithm please refer Edelsbrunner et al. [13]. The voids we compute here are the ε-stable voids.

3.4.5 Refinement using persistence

We prune the set of ε-stable voids obtained as above by computing the persistence of creators of each void and determining if it is greater than π. The pruned set of voids are (ε, π)-stable. Note that the persistence is computed with respect to the original filtration.

3.5 Correctness

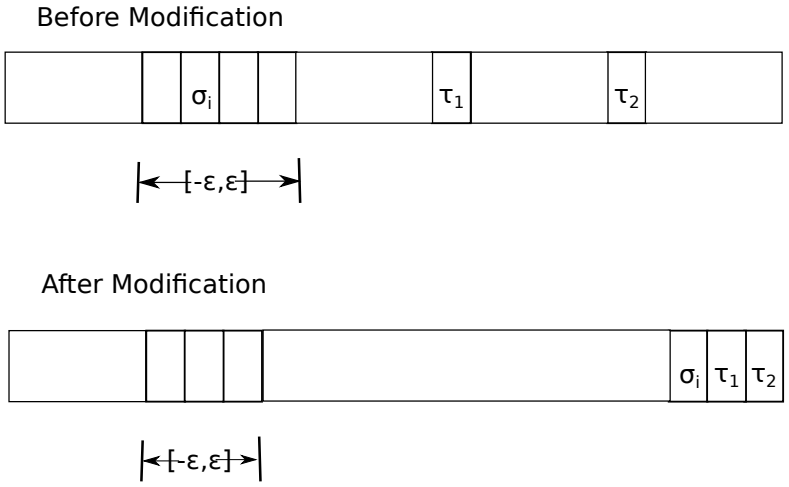


Figure 3.2: Illustration for proof of Lemma 1 and 2

Lemma 1. *Algorithm 3.3 preserves the filtration property.*

Proof. Let the alpha complex filtration be denoted by $F_{\mathcal{A}}$. We denote the filtration order by $<_F$. we have for all $\sigma_i \in F_{\mathcal{A}}$ and $\tau \in \partial(\sigma_i), \tau <_{F_{\mathcal{A}}} \sigma_i$. By definition if $\sigma_i \in \Sigma_t^{'}$ we have $dim(\sigma_i) = 2$ and if $\sigma_i \in \Sigma_T^{'}$ we have $dim(\sigma_i) = 3$. Now both Step 1 and Step 2 of Algorithm 3.3 changes the position of all simplices $\sigma_i \in \Sigma_t^{'}$ or $\sigma_i \in \Sigma_T^{'}$. We can ignore all $\sigma_i \in \mathcal{F}_{\mathcal{A}}$ where $dim(\sigma_i) = 0$ or $dim(\sigma_i) = 1$ because Steps 1,2 doesn't affect them in anyway. Any σ_i where $dim(\sigma_i) = 3$ may be moved because in a 3D complex σ_i does not have cofaces and hence Step 2 doesn't affect the filtration.

We now consider the case where $dim(\sigma_i) = 2$. We describe a proof by contradiction.

Let $\sigma_i \in \Sigma_t^{'}$ and $\mathcal{F}$ denote the family of all fitrations possible on the given pointset. This means $dim(\sigma_i) = 2$ by Algorithms 3.1 and 3.2 and it can have only two cofaces. Let τ_1, τ_2 be the cofaces of σ_i. Assume one of them, say τ_1, violates the filtration property after applying Steps 1,2. This means there exists no $F \in \mathcal{F}$ such that $\sigma_i <_F \tau_1$. But from Step 2(c).ii of Algorithm 3.2 we know that $\tau_1 \in \Sigma_T^{'}$. Also we ensure Step 1 always happens before Step 2 (refer Figure 3.2) This ensures σ_i is moved first before τ_1 and hence τ_1 is placed after σ_i. So Equation 4.9 together with the ordering of Steps 1,2 in the algorithm 3.3 implies there exists $F \in \mathcal{F}$ such that $\sigma_i <_F \tau_1$ contradicting with the earlier statement that such an F does not exist. □

Note that the ordering of Steps 1,2 is extremely important and reversing the order violates the

filtration property. Step 1 without Step 2 also violates the filtration property.

Lemma 2. *If the Algorithm mentioned in Section 3.4.4 is applied after Algorithm 3.3, then it computes the ε-stable voids.*

Proof. Let A be the alpha complex for $\alpha = 0$. Let C be the set of voids computed after applying the Algorithm 3.3. Let set of ε-stable voids be denoted by C_ε.By default we have $C_\varepsilon \subseteq C$. And we are left with only those voids which before applying the Algorithm 3.3 gets split. Let C' be that set and it members be $c'_1, c'_2 ..., c'_n$. We describe a proof by contradiction.

Let us assume that there exists a void which gets split in the interval $[-\varepsilon, \varepsilon]$ but does not belong to the set of ε-stable voids i.e.there exists $c'_i \in C'$ and $c'_i \notin C_\varepsilon$. Now c'_i gets split say m times in the interval $[-\varepsilon, \varepsilon]$. Correspondingly we will have m triangles $\{\sigma_1, ..., \sigma_m\}$ each affecting exactly one split. Each such triangle σ_i will have cofaces τ_{i1}, τ_{i2} for all $\sigma \in \{\sigma_1, ..., \sigma_n\}$ which implies that there exists τ_{i1}, τ_{i2} and $\sigma_i < \tau_{i1}, \sigma_i < \tau_{i2}$. As the triangle σ_i splits the void it means two new voids c_{ij}, c_{ik} are formed for some j, k. Now either $\tau_{i1} \in c_{ij}, \tau_{i2} \in c_{ik}$ or $\tau_{i1} \in c_{ik}, \tau_{i2} \in c_{ij}$. But Step 2 of Algorithm 3.2 adds every $\sigma \in \{\sigma_1, ..., \sigma_n\}$ to Σ'_t and adds every τs which satisfies this criteria to Σ'_T. From this we know, every such σ and τ belongs to $DT - A$ and thus c'_i remains as a single void and $c'_i \in C_\varepsilon$ contradicting the earlier statement. Hence the lemma holds. □

3.6 Stability of pockets

The above notion of stability can be extended to pockets as well. A pocket may be destroyed in the filtration by the addition of a triangle. By destroying the pocket, this triangle creates a void, or a void-pocket pair. In our current implementation, the voids that are created by the triangle are given preference over pockets that are destroyed in the $[-\varepsilon, \varepsilon]$ range. We choose to do this because a pocket with a very narrow opening (size less than ε) will not allow any ion to pass into it, and hence functions more as a void.

Therefore, the only stable pockets that are identified correspond to those that are destroyed or split into a void-pocket pair outside of the $[-\varepsilon, \varepsilon]$ range. However, in case a pocket is preferred over a stable void, the algorithm can be easily modified to delay the insertion of the triangle that destroys a pocket, along with its coface tetrahedra, to the end of the filtration.

Chapter 4

Implementation

In this chapter we discuss the implementation of the void computation and the delayed simplex insertion alogrithms. We describe the various data structures employed by the algorithm and discuss the design of the protein visualization software that we developed to view the results.

4.1 RobustVoids

We have implemented a visualization and void computation tool RobustVoids using OpenGL together with Qt for the user-interface. The GUI contains interfaces to vary α and persistence, to specify whether to display the skin surface, volumes and surface areas. It also displays the volumes and surface areas of cavities and mouths along with details like number of mouths that belong to a particular pocket. Figure 4.1 shows a screenshot of the tool.

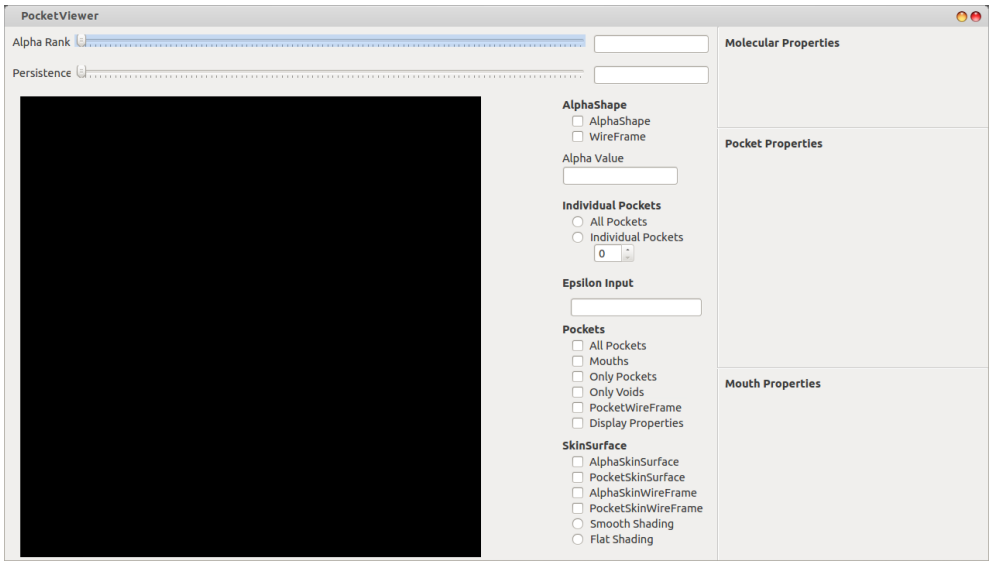


Figure 4.1: Screenshot of the viewer showing various options available

4.2 Preprocessing

The protein data is available in a standard format from the *protein data bank* (PDB format). This data contains the co-ordinates of the atoms and their Van der Waals radii. It further contains information about hetero-atoms, the sequences of amino acids present, etc. The preprocessing step generates a file that contains basic information required for further steps of the algorithm. We use ParsePDB software to preprocess the pdb input which is a part of the *proshape* distribution (Koehl et al. [19]) .

4.3 Delaunay triangulation

The result of the preprocessing step is used to generate the Delaunay triangulation (referred to as *DT* henceforth). We use an incremental algorithm to construct the Delaunay triangulation (for details see Koehl et al. [19]). *Simulation of Simplicity* (SoS) (refer Edelsbrunner and Mücke [10] for details) is used to handle degenerate cases and ensure *general position*. We have re-implemented the Delaunay triangulation routines in C++ to utilise dynamic memory management and the Standard Template Libraries (STL). The triangulation is stored as a list of tetrahedra. Triangles and edges are extracted from this list.

We use standard data structures from the C++ STL to store vertices, edges, triangles and tetrahedra. While the vertices store coordinates and radii, higher order simplices store indices of vertices that form the simplices. Additionally tetrahedra store indices to their neighbours, boundaries and flow relations

(see Section 4.5) and triangles store their co-faces to ensure faster access. We also store the normals of triangles for rendering purposes.

4.4 Alpha Shape Spectrum

Once the tetrahedra, triangles, edges are defined as discussed in Section 4.3 we build the alpha shape spectrum as discussed in Edelsbrunner [6]. We compute the size of each simplex in the DT as defined in Section 2.4. We use the *GNU Multi-Precision* (GMP) library to avoid numerical inconsistencies. Alpha shape spectrum is stored in ascending order of α in a master list which contains the indices and type of the simplex inserted into the filtration at the value α. Once the shape spectrum is generated we can extract/build alpha complexes for any given value α as a prefix of the spectrum.

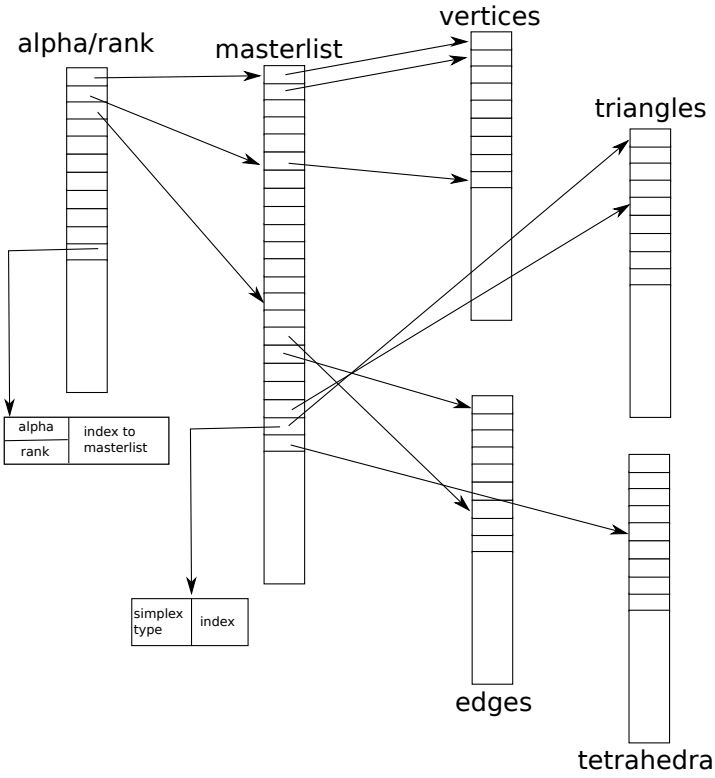


Figure 4.2: Representation of the data structures used in the alpha shape computation.

In Figure 4.2 we see various data structures utilised in computation of alpha shapes. We store the alpha values in an ascending order and the corresponding ranks as an array in which each entry has an index to the masterlist. The masterlist is stored as a dynamic array (using STL vector) in which each entry contains two fields, an index into the data structures which stores simplices (as discussed in 4.3) and the type of simplex pointed to by the index. Using this we can efficiently compute the set

of simplices comprising the α-complex for a given value of α. Figure 4.3 shows few examples of alpha shapes rendered in various modes.

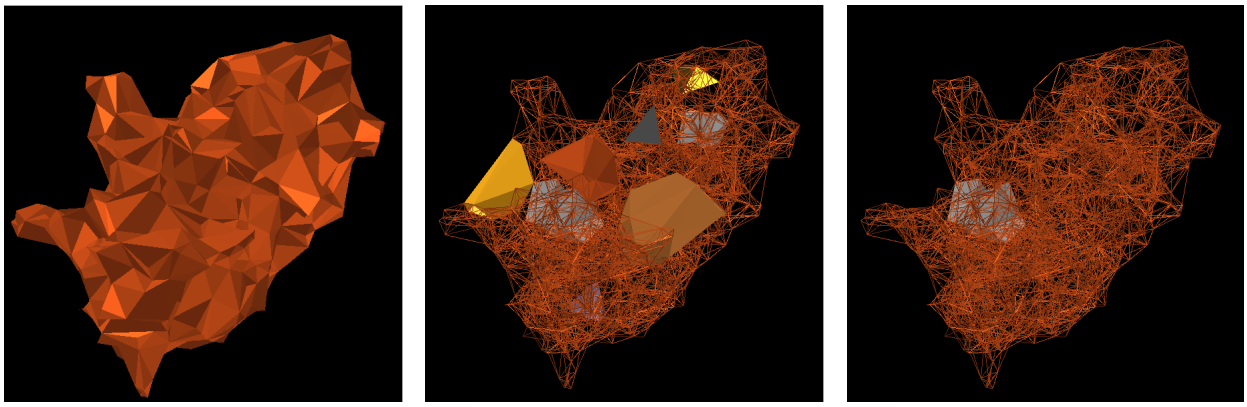


Figure 4.3: **Left:** Alphashape rendering in solid mode for protein Rnase s [PDB ID: 1d5h] for alpha value 0. **Center:** in wireframe mode **Right:** showing only voids

4.5 Flow and Depth Computation

We compute flow and depth (defined in Section 2.6) which is necessary for cavity computation as discussed in Edelsbrunner et al. [13]. Assuming the presence of a filtration, for each tetrahedron σ we have four cases. The cases depend on whether the orthogonal center y lies inside σ or outside and if it lies outside, whether it sees 1,2 or 3 triangles of the tetrahedron σ. If y lies outside σ, for each triangle visible from y we define $\sigma \prec \tau$ where τ is tetrahedron on the other side of the triangle. We introduce a dummy tetrahedron ω representing the space surrounding the molecule. By definition, its orthogonal center is at ∞, so ω can only be a successor of other tetrahedra. It is called the sink of the relation and the other sinks are tetrahedra that contain their orthogonal centers. We compute the ancestor set which is in turn used to compute cavities.

4.6 Cavities and Mouth Computation

First we re-adjust the radii of the atoms using the current α value, a step that is necessary for the computation of cavities. The flow and depth computation results are used to find the cavities in the following manner. Cavities are defined by the tetrahedra that neither belong to the complex nor to the ancestor set of ω. We collect tetrahedra in $DT - K - Anc\omega$ and partition this collection into components using a union-find data structure. Mouths can be computed as the intersction of convex-hull (CH) and the faces of the above collection of tetrahedra. We collect the boundary triangles that do not belong

to K i.e $CH \ - \partial(K)$ and again use the union-find data structure to partition them into connected components.

Cavities are stored as a set of indices of tetrahedra while mouths are similarly stored as a set of indices of triangles. Additionally a mapping between cavities and mouths are stored. Voids and pockets are distinguished based on the number of mouths. Figure 4.4 shows cavities extracted for the protein molecule Rnase s [PDB ID: 1d5h].

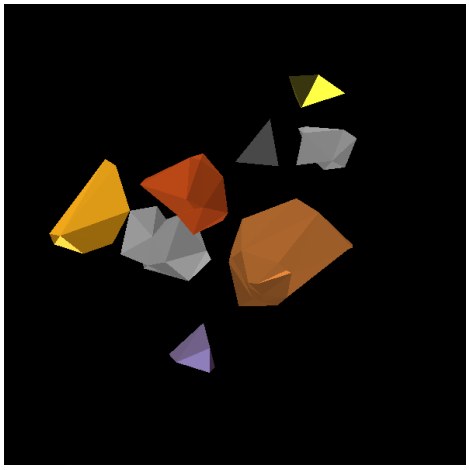


Figure 4.4: Voids and pockets in protein Rnase s [PDB ID: 1d5h]

4.7 Volume and Surface Area computations

Volume and surface area are computed using inclusion-exclusion formulas discussed by Liang et al. [23, 24], Edelsbrunner and Fu [8]. We implement primitives to compute volumes and surface areas of a set of building blocks including spheres, intersections of k spheres ($k = 2, 3, 4$), sectors, wedges and pawns. These primitives are used in the inclusion-exclusion formulas to determine the total volumes and surface areas. Figure 4.5 shows the volume and surface area computed for protein Rnase s [PDB ID: 1d5h] for $\alpha = 0$ and displayed within our viewer.

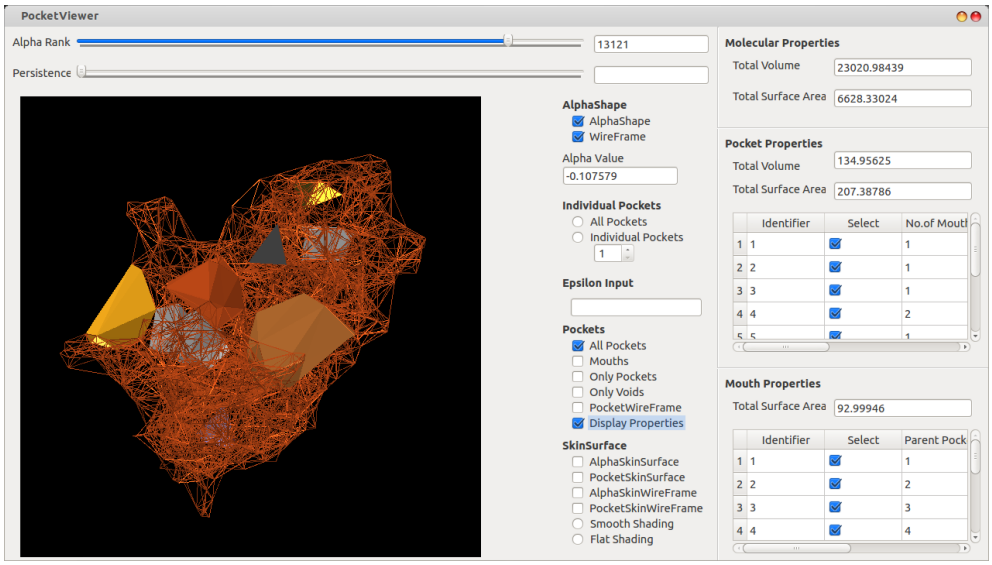


Figure 4.5: Volume and surface area displayed within our viewer.

4.8 Skin Surface computation

We compute skin surfaces of the current α-shape and the cavities using software developed by Cheng and Shi [3]. We compute the surface on demand because computing the skin surface is a costly operation and may takes several seconds to a few minutes. The current implementation of skin surfaces works only for a single connected component and hence we use the interface multiple times, once for each component of the cavity to compute its surface. We store the output generated by the skin surface software as a set of triangles and vertices. We calculate the vertex normals along with the face normals to ensure smooth shading.Figure 4.6 shows the skin surface for protein Rnase s [PDB ID: 1d5h] for $\alpha = 0$ along with its pockets.

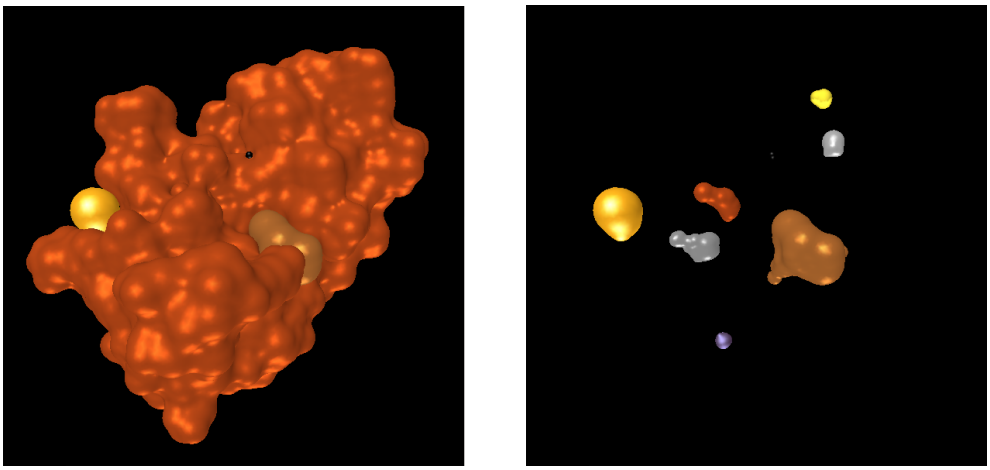


Figure 4.6: **Left:** Skin surface of protein Rnase s [PDB ID: 1d5h] along with its pockets for $\alpha = 0$. **Right:** Skin surface of the cavities of protein Rnase s [PDB ID: 1d5h] for $\alpha = 0$.

4.9 Persistence computation

We utilise the filtration obtained in Section 4.4 to pair simplices and compute persistence. In particular we compute the persistence of $d-1$ dimensional simplices, which corresponds to persistence of cavities.

4.10 Modifying the Filtration

We obtain ε as input from the user. Algorithm 3.1 computes the rank corresponding to the given ε value from the alpha shape spectrum. Next we create different lists of simplices based on their dimensions and send them as input to Algorithm 3.2. We compute the cavities for the range of ranks computed by Algorithm 3.1 and refine the lists of triangles and tetrahedra to generate the candidate set which contains the simplices which are to be moved out of the ε interval. We employ a validity flag to mark the simplices to be moved outside the interval of interest. See Figure 4.7. The validity flag also helps in updating the radii of the atoms corresponding to the simplices moved out of the interval of interest, thus helping us to track changes in volumes and surface areas. We recompute the cavities as discussed in Section 4.6 after the filtration modification to arrive at the set of (ε, π)-stable voids as required.

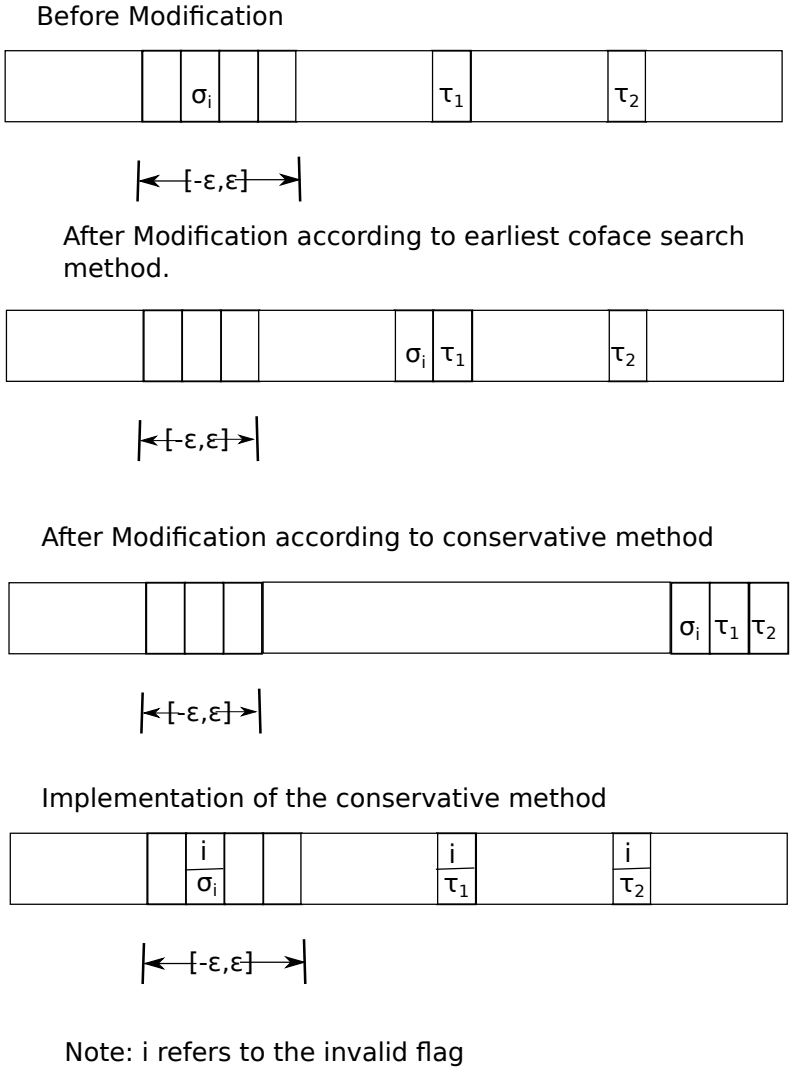


Figure 4.7: The effect of filtration modification on the master list

4.10.1 Complexity

The running time of Algorithm 3.1 and hence the whole algorithm depends on the input ε, Step 3 of Algorithm 3.1 is a loop over a set of consecutive ranks and in the worst case scenario traverses the entire alpha shape spectrum. So the running time is $O(n^2)$, where $n = |S|$ is the number of atoms. Algorithm 3.2 loops through the same set of ranks and computes the cavities for all ranks. Cavity computation requires the union-find algorithm on the set of simplices which has an amortised cost of $O(\alpha(m))$ where m is the number of simplices and $\alpha(m)$ represents the inverse of the Ackermann function. So the total cost in the worst case is $O(m * \alpha(m))$. Algorithm 3.3 performs the actual modification in linear time

on the number of simplices m which has the worst case complexity of $O(n^2)$. Algorithms 3.1 and 3.2 can be combined during the implementation to speed-up the running time of the algorithm. The total running time is $O(m * \alpha(m) + m)$. where m is the number of simplices. In practice ε is a small real number and we almost never encounter the worst case scenario.

Chapter 5

Results

In this chapter we will discuss the experiments which we have done using various protein datasets and the results we obtain by running the algorithm on those datasets.

5.1 Data

Protein datasets are obtained from Berman et al. [1], RCSB [32] which is an online repository that stores proteins in the standard format .pdb. We perform experiments on a set of proteins and have tabulated detailed results for the proteins with PDB ID 2CI2, 4HHB and 4B87.

5.2 Visualization of stable cavities

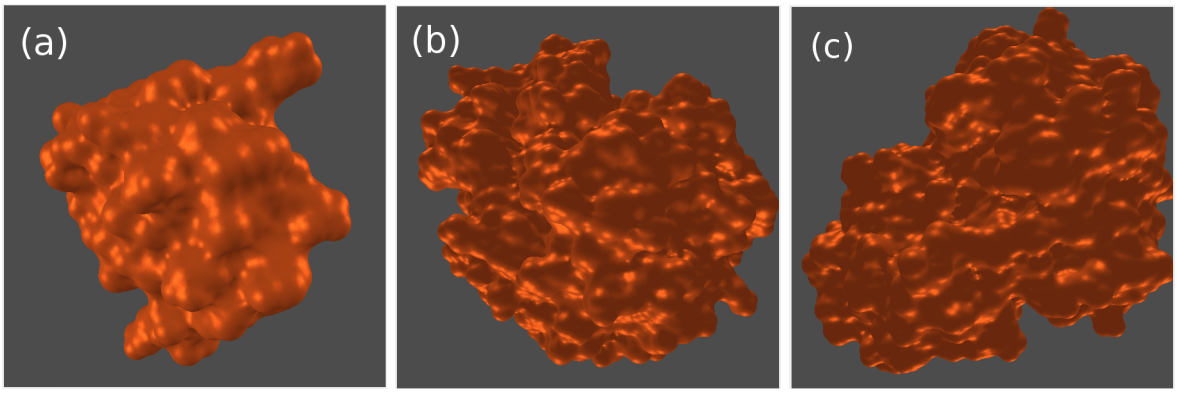


Figure 5.1: Visualization of the skin surface of the proteins used in our experiments. **(a)** 2CI2. **(b)** 4HHB. **(c)** 4B87.

Figure 5.1 shows molecules rendered using our software for proteins having PDB identifiers 2CI2, 4HHB and 4B87.

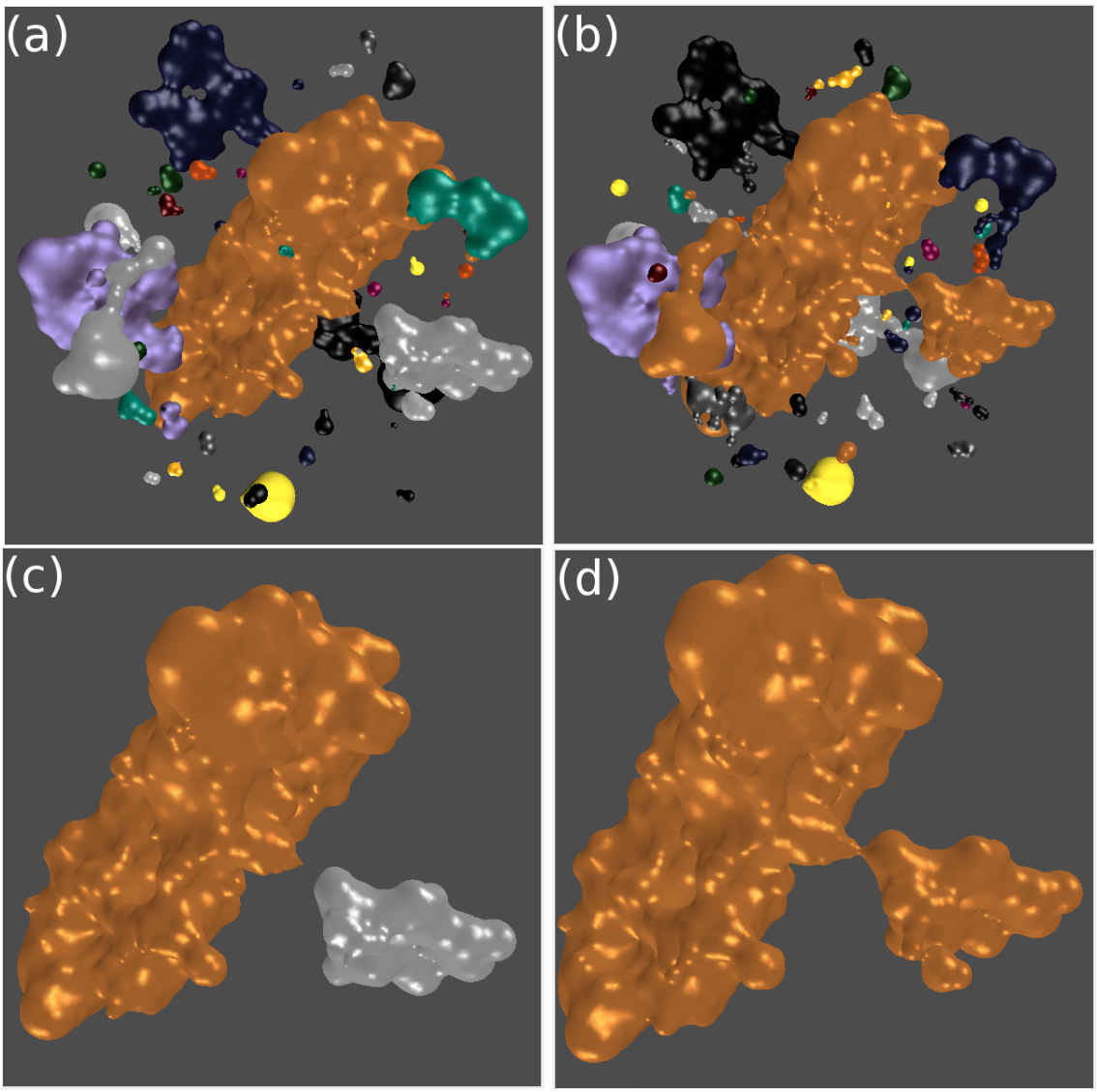


Figure 5.2: Visualization of voids of the protein 4HHB. The values $\varepsilon = 1.0$ and $\pi = 0.01$ was used to compute the set of stable voids. **(a)** Voids in the volume computed at $\alpha = 0$. **(b)** The set of $(1, 0.01)$-stable voids. **(c)** Two nearby voids in the protein. **(d)** These two voids merge together resulting in a single stable void.

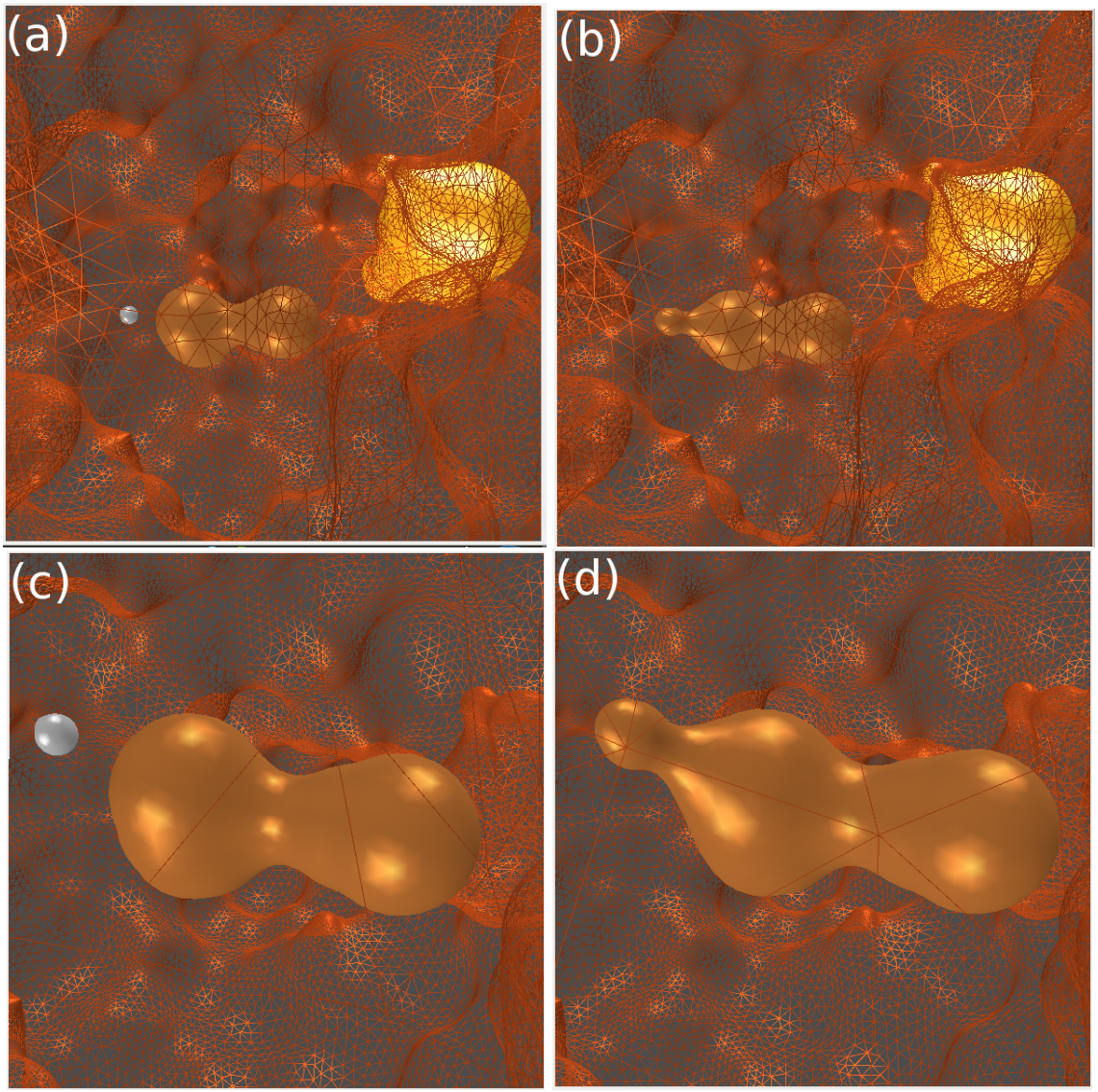


Figure 5.3: Visualization of voids of the protein 2CI2. The values $\varepsilon = 0.3$ and $\pi = 0.01$ was used to compute the set of stable voids. **(a)** Voids in the volume computed at $\alpha = 0$. Number of voids = 3. **(b)** The set of $(0.3, 0.01)$-stable voids. Number of $(0.3, 0.01)$-stable voids = 2. **(c)** Two nearby voids in the protein rendered solid and the skin surface of the molecule rendered in wireframe mode. **(d)** The merged stable void shown along with the skin surface of the molecule.

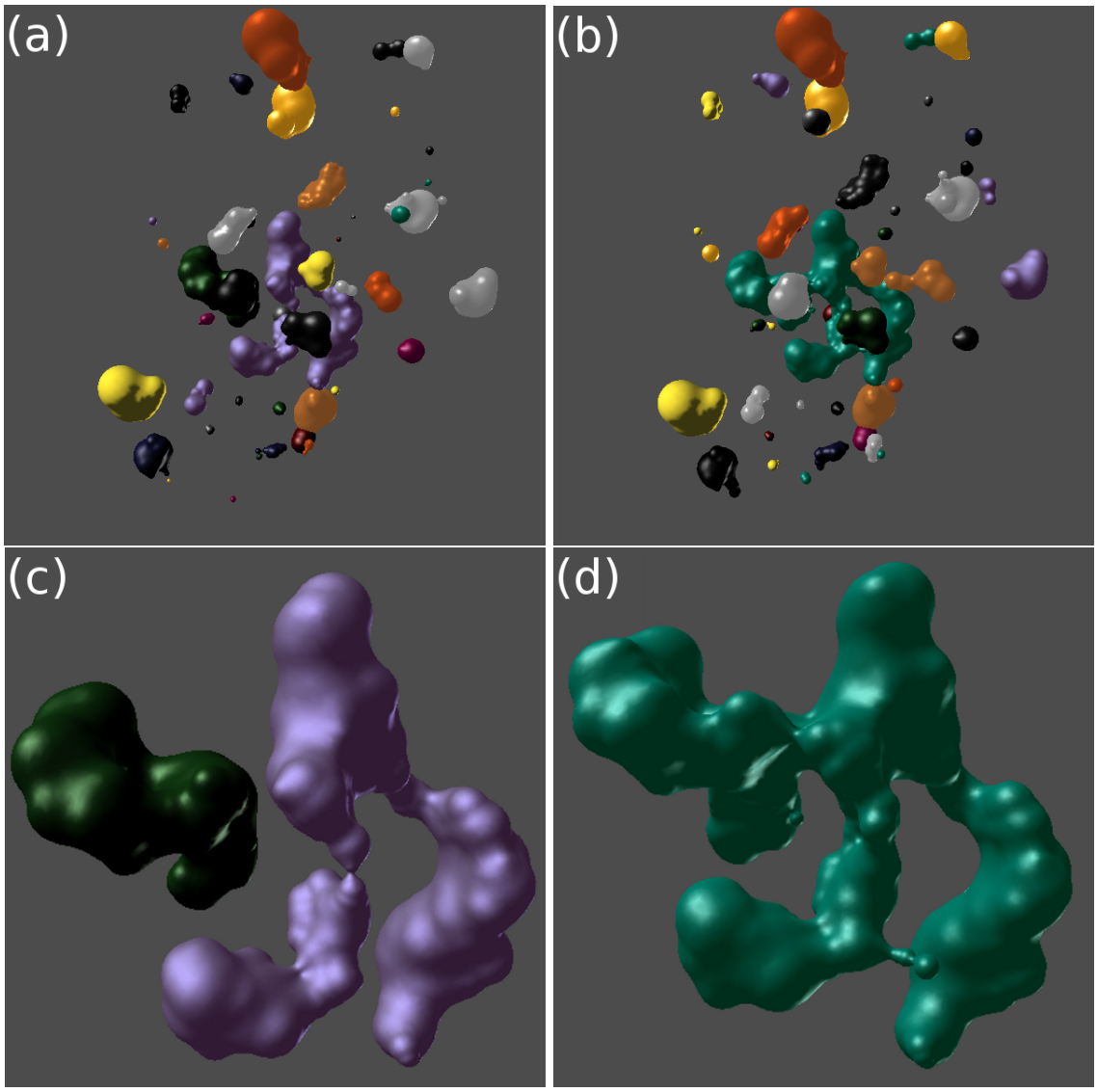


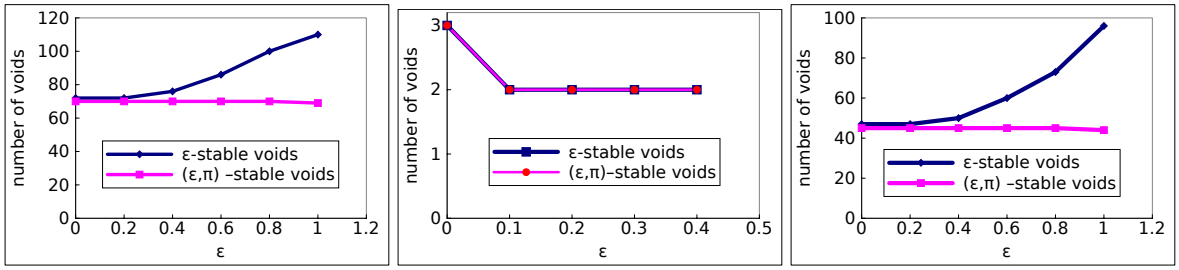
Figure 5.4: Visualization of voids of the protein 4B87. The values $\varepsilon = 1.0$ and $\pi = 0.01$ was used to compute the set of stable voids. **(a)** Voids in the volume computed at $\alpha = 0$. Number of voids = 47. **(b)** The set of $(1.0, 0.01)$-stable voids. Number of $(1.0, 0.01)$-stable voids = 44. **(c)** Two of the nearby voids in the protein. **(d)** These voids merge together resulting in a single stable void.

The protein 4HHB has a total of of 72 voids, shown in Figure 5.2(a). Using a value of $\varepsilon = 1.0$ and further removal of voids having persistence $\pi < 0.01$ results in a total of 70 $(1.0, 0.01)$-stable voids, see Figure 5.2(b). Figure 5.2(c) shows two nearby voids in the protein which merge to form a single stable void, see Figure 5.2(d). Note that a value of $\varepsilon = 1.0$ is equivalent to an increase / decrease of the radius of an atom by at most 0.33Å. This quantity is within the tolerated 0.5Å used by the biologists.

Figure 5.5(a) plots the number of (ε, π)-stable voids for various values of ε. Note that, when increasing

the value of ε, we are considering voids from a larger α range. This could potentially increase the number of ε-stable voids reported. However, such voids usually have low persistence and are therefore not (ε, π)-stable. We use a constant value of $\pi = 0.01$ in all our experiments.

We repeated the above experiment for the proteins 2CI2 and 4B87. Figures 5.3 and 5.4 show the visualization of the (ε, π)-stable voids for these proteins. The variation of the number of such voids with ε is shown in Figure 5.5.



(a) Protein 4HHB (b) Protein 2CI2 (c) Protein 4B87

Figure 5.5: Graphs showing the variation of the number of voids with varying ε. Note that there is an increase in ε-stable voids as we consider a larger interval but (ε, π) voids are less than or equal to original number of voids.

5.3 Properties of stable cavities

In addition to identifying the set of stable cavities in a protein molecule, we are also interested in the robust computation of volumes of these cavities. The volumes are then used in the energy calculations and analysis of packing of the protein. In this section, we first describe our volume computation strategy, followed by analysing the effect of stability on the volume of the cavities.

Computing correct volumes of cavities. There is usually a variation in the volumes of cavities computed using various methods. This variation arises due to the different models used for computing the volumes. In order to facilitate the comparison of volumes computed using our software and existing tools used by biologists, we perform an additional normalization of the computed volume as specified in Chakravarty et al. [2]. This is accomplished by using model mutants to create artificial voids. The volumes of these voids are then computed. The difference between the computed volume and the actual volume of these artificial voids (obtained from Richards [33]) is used to obtain the required normalization.

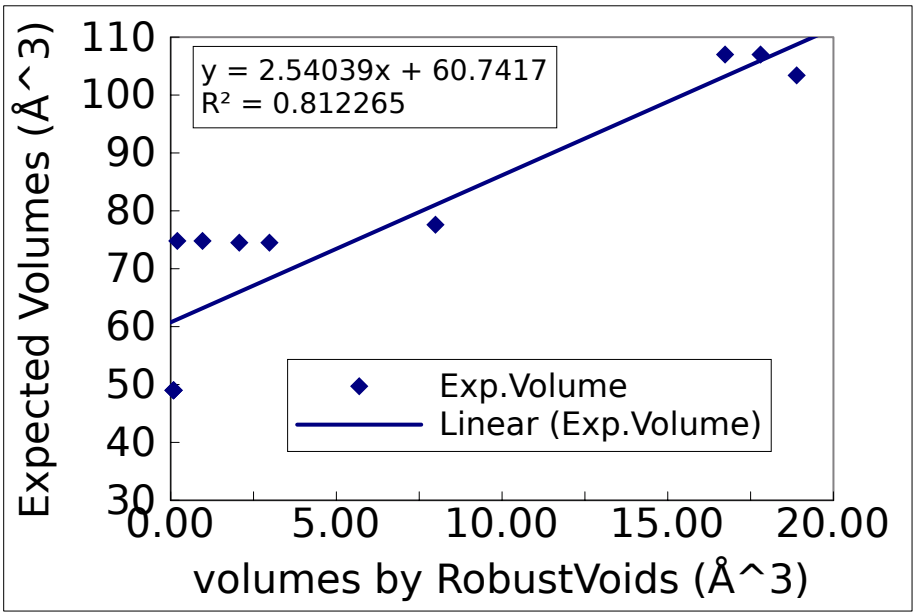


Figure 5.6: Plot of the computed and actual volumes of the artificial voids that were generated using mutant models. This plot was used to obtain the linear normalization function for our software, depicted using the blue line.

We used 13 different model mutants to create a set of artificial voids. The resulting volumes were then used to compute the linear normalization function, see Figure 5.6. The coefficients for the best fit line are

$$Volume = 2.54 \times ComputedVolume + 60.77$$

In order to verify the correctness of the volumes computed by our software, we compare them to the volumes computed by an existing software called *MC Cavity* implemented in Chakravarty et al. [2], see Figure 5.7. Note that we use normalised volumes in this graph.

Effect of stability on the volume of cavities. We compute the volumes of the set of (ε, π)-stable voids for proteins 4HHB, 4B87 and 2CI2, respectively. Figures 5.8 plots the total volume of all voids for varying ε for these three molecules. Note that the total volume of all stable voids increases with increasing ε. This is due to the decrease in the radii of the atoms, which contributes to an increase in the volume of voids. However this increase is marginal ($< 1\%$) given the small values of ε. The merging of two nearby voids into a single stable void does not effect the total volume. However, volumes of individual voids could change drastically. For example, consider the two large nearby voids in Figure 5.2(c). They merge to form a single (ε, π)-stable void as shown in Figure 5.2(d). The volume of this stable void is approximately equal to the sum of the volumes of the two nearby voids.

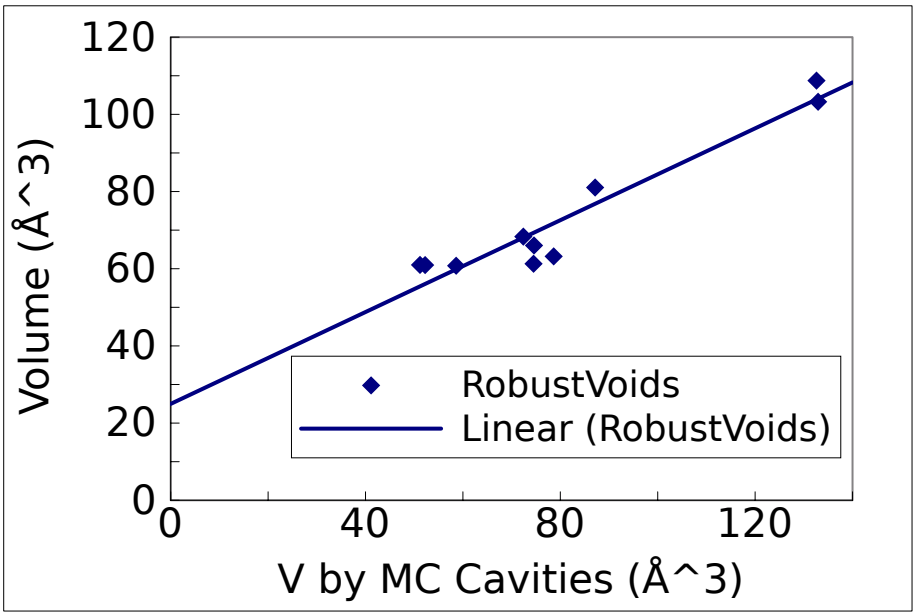
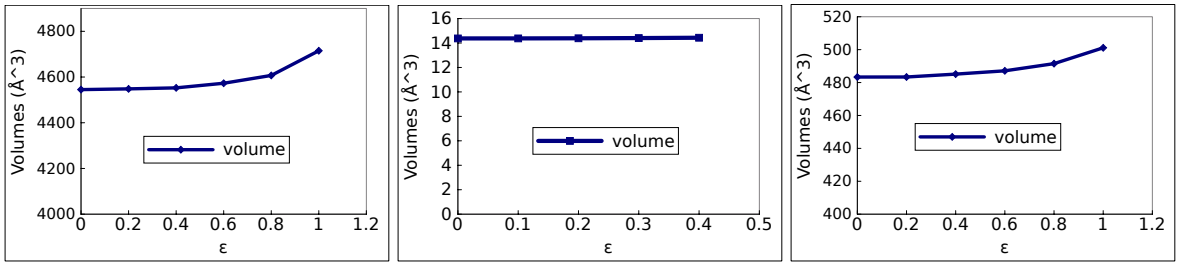


Figure 5.7: Graph showing the relationship between normalised volumes calculated by RobustVoids and MC Cavity



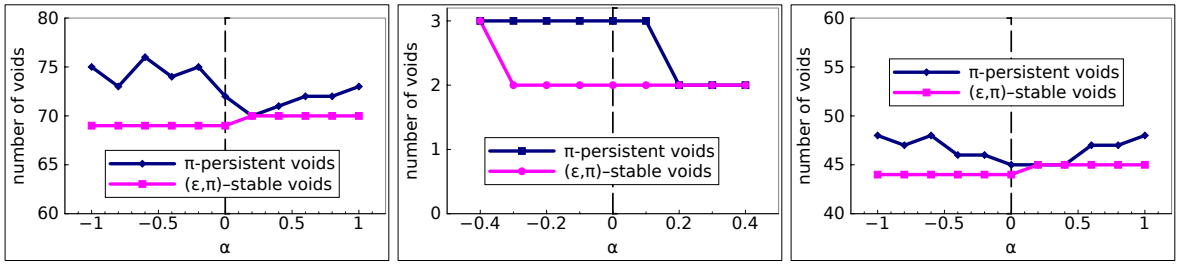
(a) Protein 4HHB (b) Protein 2CI2 (c) Protein 4B87

Figure 5.8: Graphs showing the variation of the total volume of voids with varying ε.

Robustness of (ε, π)-stable cavities. Figure 5.9 plots the number of (ε, π)-stable voids and π-persistent voids for various values of α for $\varepsilon = 1.0$ in case of 4HHB and 4B87 and $\varepsilon = 0.3$ in case of 2CI2. The number of (ε, π)-stable voids remain constant over an interval as the method considers all the creation/destruction events happening in the interval $[-\varepsilon, \varepsilon]$ for the computation. But the number of π-persistent voids vary as they are sensitive to creation/destruction of voids at a particular α value.

5.4 Observations

We do not observe a monotonic behavior in the number of ε-stable cavities. This is because the filtration modification may result in an increase in the number of cavities. This may seem counter-intuitive but it can be explained by the fact that the count includes those cavities that get created within the ε-interval



(a) Protein 4HHB (b) Protein 2CI2 (c) Protein 4B87

Figure 5.9: Graphs showing the variation of the total volume of voids for constant ε and varying α.

and may have small lifetimes. We see the same behaviour reflected in the calculation of the cavity volumes because the cavities with small lifetimes are also considered for the volume calculation. It is more appropriate to consider the (ε, π)-stable cavities. Here we observe that refining the set of cavities with a persistence threshold removes cavities with small or insignificant lifetimes and gives us a truer picture of the number of cavities. Similarly we see that the volume of the cavities also reflect the change in the number of cavities after the refinement. But the change in the volume is not very significant as the cavities that have insignificant lifetimes also tend to have small volumes and hence removing them does not result in huge changes in the total volume. We also observe that running time is heavily dependent on ε because it determines the range of ranks that are processed in the alpha shape spectrum. Refining the set of candidate simplices requires us to calculate cavities for those ranks. Thus the running time is impacted by the time required to calculate the cavities too.

Chapter 6

Conclusion and Future Work

Voids and pockets in a protein molecule play an important role in determining its functionality and stability. Inaccuracies in the empirically determined radii of the atoms leads to an incorrect estimation of these cavities and their properties. In this work, we defined a novel notion of robust cavities that is insensitive to the perturbation of the atomic radii. We also proposed a topological framework to compute these robust cavities.

From the biologist's point of view, obtaining a stable protein is the starting point of many applications including studies of binding and interactions. Surface pockets often form part of the active site of proteins, whereas internal voids are often relevant structurally as features that affect the overall stability of the protein. It is established that filling up internal voids improves the packing of the protein thus increasing stability. In this respect detecting structurally robust cavities inside the protein and computing their volumes helps determine which mutations to perform to improve internal packing and obtain a stable protein. Our software while detecting robust cavities and calculating volumes, also provides an interactive framework to explore the protein and its cavities. The biologist may use their domain knowledge and discretion, to decide which cavities affect the proteins function and which mutations would be required for their specific applications.

The method addresses the inaccuracies in the measurements of the radii by selectively varying the radii for a specific set of atoms. But the positional uncertainties which arise due to the motion of the molecules is not addressed. The method assumes the union of balls representation. We have been able to present some visual evidence that slight perturbations in the radii results in a larger cavity instead of smaller cavities. The value of ε is lower than the typical experimental error in crystallographic measurements. We plan to further investigate the relationship between the perturbation in the atom radii corresponding to the delayed simplex insertion and the structural and functional properties of the protein. Future work also includes generalizing the framework to use empirically determined intervals of radii for each atom type and addressing the issue of biological implications of the method.

Bibliography

[1] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, TN Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The Protein Data Bank. *Nucleic acids research*, 28(1):235–242, 2000. 5.1

[2] S. Chakravarty, A. Bhinge, and R. Varadarajan. A procedure for detection and quantitation of cavity volumes in proteins. *Journal of Biological Chemistry*, 277(35):31345–31353, 2002. 1.4, 2.7, 5.3, 5.3

[3] H.L. Cheng and X. Shi. Quality mesh generation for molecular skin surfaces using restricted union of balls. *Computational Geometry*, 42(3):196–206, 2009. 1.4, 4.8

[4] C.J.A. Delfinado and H. Edelsbrunner. An incremental algorithm for betti numbers of simplicial complexes. In *Proc. Symposium on Computational geometry*, pages 232–239, 1993. 1.4, 2.5, 2.5

[5] J. Dundas, Z. Ouyang, J. Tseng, A. Binkowski, Y. Turpaz, and J. Liang. CASTp: computed atlas of surface topography of proteins with structural and topographical mapping of functionally annotated residues. *Nucleic acids research*, 34(2):W116–W118, 2006. 1.4

[6] H. Edelsbrunner. *Weighted alpha shapes*. University of Illinois at Urbana-Champaign, Department of Computer Science, 1992. 1.3, 1.4, 2.4, 4.4

[7] H. Edelsbrunner. The union of balls and its dual shape. *Discrete & Computational Geometry*, 13 (1):415–440, 1995. 1.4

[8] H. Edelsbrunner and P. Fu. Measuring space filling diagrams and voids. Technical report, UIUC-BI-MB-94-01, Beckman Inst., Univ. Illinois, Urbana, Illinois, 1994. 1.4, 2.7, 4.7

[9] H. Edelsbrunner and P. Koehl. The geometry of biomolecular solvation. *Combinatorial & Computational Geometry*, 52:243–275, 2005. 1.4

[10] H. Edelsbrunner and E.P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (TOG)*, 9(1):66–104, 1990. 4.3

[11] H. Edelsbrunner and E.P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, 1994. 1.4, 2.4

[12] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983. 1.4, 2.3, 2.4

[13] H. Edelsbrunner, M. Facello, and J. Liang. On the definition and the construction of pockets in macromolecules. *Discrete Applied Mathematics*, 88(1):83–102, 1998. 2.6, 2.7, 3.4.4, 4.5

[14] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002. 1.3, 1.4, 3.1.3

[15] K.D. Gibson and H.A. Scheraga. Exact calculation of the volume and surface area of fused hard-sphere molecules with unequal atomic radii. *Molecular Physics*, 62(5):1247–1265, 1987. 1.4

[16] Deok-Soo Kim and K Sugihara. Tunnels and voids in molecules via voronoi diagram. In *Proc. Symp. Voronoi Diagrams in Science and Engineering (ISVD)*, pages 138–143, 2012. 1.4

[17] Deok-Soo Kim, Youngsong Cho, Kokichi Sugihara, Joonghyun Ryu, and Donguk Kim. Three-dimensional beta-shapes and beta-complexes via quasi-triangulation. *Computer-Aided Design*, 42 (10):911–929, 2010. 1.4

[18] Deok-Soo Kim, Joonghyun Ryu, Hayong Shin, and Youngsong Cho. Beta-decomposition for the volume and area of the union of three-dimensional balls and their offsets. *Journal of Computational Chemistry*, 2012. 1.4

[19] P. Koehl, M. Levitt, and H. Edelsbrunner. Proshape: understanding the shape of protein structures. *Software at biogeometry.duke.edu/software/proshape*, 2004. 1.4, 4.2, 4.3

[20] M. Krone, M. Falk, S. Rehm, J. Pleiss, and T. Ertl. Interactive exploration of protein cavities. In *Computer Graphics Forum*, volume 30, pages 673–682, 2011. 1.4

[21] Michael Krone, Guido Reina, Christoph Schulz, Tobias Kulschewski, Jürgen Pleiss, and Thomas Ertl. Interactive extraction and tracking of biomolecular surface features. In *Computer Graphics Forum*, volume 32, pages 331–340. Wiley Online Library, 2013. 1.4

[22] B. Lee and F.M. Richards. The interpretation of protein structures. *Journal of Molecular Biology*, 55(3):379–400, 1971. 1.4, 2.1

[23] J. Liang, H. Edelsbrunner, P. Fu, P.V. Sudhakar, and S. Subramaniam. Analytical shape computation of macromolecules: I. molecular area and volume through alpha shape. *Proteins Structure Function and Genetics*, 33(1):1–17, 1998. 1.4, 2.7, 4.7

[24] J. Liang, H. Edelsbrunner, P. Fu, P.V. Sudhakar, and S. Subramaniam. Analytical shape computation of macromolecules: II. inaccessible cavities in proteins. *Proteins Structure Function and Genetics*, 33(1):18–29, 1998. 1.4, 2.7, 4.7

[25] J. Liang, H. Edelsbrunner, and C. Woodward. Anatomy of protein pockets and cavities. *Protein Science*, 7(9):1884–1897, 1998. 1.4, 2.7

[26] N. Lindow, D. Baum, and H.C. Hege. Voronoi-based extraction and visualization of molecular paths. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2025–2034, 2011. 1.4

[27] N. Lindow, D. Baum, A.N. Bondar, and H.C. Hege. Dynamic channels in biomolecular systems: Path analysis and visualization. In *Proc. IEEE Symposium on Biological Data Visualization (BioVis)*, pages 99–106, 2012. 1.4

[28] J.R. Munkres. *Elements of Algebraic Topology*, volume 2. Addison-Wesley Menlo Park, CA, 1984. 2.2

[29] R. Pavani and G. Ranghino. A method to compute the volume of a molecule. *Computers & Chemistry*, 6(3):133–135, 1982. 1.4

[30] M. Petřek, P. Košinová, J. Koča, and M. Otyepka. MOLE: A Voronoi diagram-based explorer of molecular channels, pores, and tunnels. *Structure*, 15(11):1357–1363, 2007. 1.4

[31] Martin Petřek, Michal Otyepka, Pavel Banáš, Pavlína Košinová, Jaroslav Koča, and Jiří Damborskỳ. Caver: a new tool to explore routes from protein clefts, pockets and cavities. *BMC bioinformatics*, 7(1):316, 2006. 1.4

[32] RCSB. RCSB Protein Data Bank. URL `http://www.rcsb.org/pdb/home`. 5.1

[33] F.M. Richards. Areas, volumes, packing, and protein structure. *Annual Review of Biophysics & Bioengineering*, 6:151–76, 1977. 5.3

[34] Mirco S Till and G Matthias Ullmann. Mcvol-a program for calculating protein volumes and identifying cavities by a monte carlo algorithm. *Journal of molecular modeling*, 16(3):419–429, 2010. 1.4

[35] Xiaoyu Zhang and Chandrajit Bajaj. Extraction, quantification and visualization of protein pockets. In *Proc. Computer System Bioinformatics Conference*, volume 6, pages 275–286, 2007. 1.4