Geometric and Topological Methods for Biomolecular Visualization

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE

Faculty of Engineering

BY

Talha Bin Masood

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

March, 2018

Declaration of Originality

I, **Talha Bin Masood**, with SR No. **04-04-00-10-12-12-1-09647** hereby declare that the material presented in the thesis titled

Geometric and Topological Methods for Biomolecular Visualization

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2012-2017**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Prof. Vijay Natarajan Advisor Signature

DEDICATED TO

Ammi, Abbu, Aapi and Hamza

Acknowledgements

First of all, I would like to thank my family for their patience and support. They provided encouragement when I needed it. I could depend on them for motivation and inspiration in the times of my frustration.

I would like to thank my thesis advisor, Prof. Vijay Natarajan, for his excellent guidance. I am thankful to him for introducing me to the exciting fields of scientific visualization and computational topology. I would like to sincerely acknowledge his invaluable support in all forms throughout my Ph.D. I am deeply indebted to him for his incredible tolerance, understanding and support during the last phase of my Ph.D. I am also grateful to him for providing me multiple collaboration opportunities during this work.

I consider myself fortunate to have worked with many wonderful researchers. I had the opportunity to interact with Prof. Nagasuma Chandra and Sandhya Sankaran during the ChExVis project, to whom I am incredibly grateful for their useful suggestions and advice. I worked with Prof. Raghavan Varadarajan, Siddharth Patel, Raghavendra Sridharamurthy and Harish Doraiswamy on the problem of extracting robust cavities; I am exceptionally grateful to all of them. For Facet-JFA algorithm, I am thankful to Hari Krishna Malladi for his useful insight which led to this result. I thank Prof. Sathish Vadhiyar, Dr. Tathagata Ray and Nikhil Ranjanikar for multiple fruitful discussions while designing the parallel algorithm for alpha complex. Lastly, I would like to thank Abhishek Rathod, a brilliant theoretician and a kind friend, with whom I worked on several problems in Morse theory.

VGL at CSA was my second home during the last six years. I came in contact with many wonderful people through this lab. I learned a lot about research from my peers. Especially, I looked up to Harish, Dilip and Nithin, and got inspiration from their research work. I enjoyed the time I spent with my lab mates discussing different topics. Abhishek, Giri, Tarun, Akash, Nagarjun, Raghavendra and Adhitya deserve special mention for providing great company. I would also like to thank my friends from outside VGL, Gaurav and Santanu, for making my early days at IISc fun and enjoyable.

I was supported by Microsoft Corporation and Microsoft Research India under the Microsoft

Research India Ph.D. Fellowship Award. This work was partially supported by the Department of Science and Technology, India, under Grant SR/S3/EECE/0086/2012 and the DST Center for Mathematical Biology, IISc, under grant SR/S4/MS:799/12.

Abstract

Biomolecules like proteins are the basic building blocks of living systems. It has been observed that the structure of a biomolecule plays an important role in defining its function. In this thesis, we describe novel geometric and topological techniques to understand the structure of molecules. In particular, we focus on the problems related to identification and visualization of cavities and channels in proteins. Cavities refer to empty regions within the molecule, while channels are pathways through the cavities. We pursue an integrated geometric and topological approach towards solving the problems in this domain. While topological structures provide efficient data structure representations of molecular space, geometric techniques allow accurate computation of various geometric measures having biological significance.

In the first part of the thesis, we describe two methods: one for extraction and visualization of biomolecular channels, and the other for extraction of cavities in uncertain data. We also describe the two software tools based on the proposed methods targeted at the end-user, the biologists. These two web server tools publicly available for use are called ChExVis and RobustCavities.

The first method uses an alpha complex based framework for extraction and visualization of geometrically feasible channels in biomolecules. We show that our proposed method has several advantages in terms of representation power over existing channel finding algorithms. In addition, we present novel ways of visualizing the amino-acids lining the channels together with their physicochemical properties. The second method addresses the problem of cavity extraction in biomolecules while taking into account uncertainties associated with empirically determined atomic positions and radii. We propose an approach that connects user-specified cavities by computing an optimal conduit within the region occupied by the molecule. The conduit is computed using a topological representation of the occupied and empty regions and is guaranteed to satisfy well defined geometric optimality criteria. We also describe a user interface with multiple linked views for interactive extraction and exploration of stable cavities. We demonstrate the utility of both the proposed methods using multiple case studies.

In the second part of the thesis, we describe efficient parallel algorithms for two geometric structures widely used in the study of biomolecules. One of the structures we discuss is discrete Voronoi

diagram which finds applications in channel visualization, while the other structure is alpha complex which is extremely useful in studying geometric and topological properties of biomolecules.

We introduce a variant of the jump flooding algorithm to compute the discrete Voronoi diagram called FACET-JFA. The algorithm optimizes the number of pixels processed by computing only the faces of the Voronoi tessellation. We observed speed-up of upto 10x over JFA. As an application of the proposed algorithm, we present a GPU based method for extraction of channel centerlines in biomolecules. Secondly, we propose a GPU based parallel algorithm for the computation of the alpha complex, a subcomplex of the Delaunay triangulation that is widely used to represent biomolecules. The algorithm exploits the knowledge of typical distribution and sizes of atoms in biomolecules. Practically, we observed speed-up of upto 22x over the state-of-the-art algorithm using our implementation.

Publications based on this Thesis

1. Talha Bin Masood, Sankaran Sandhya, Nagasuma Chandra, and Vijay Natarajan. "ChExVis: A tool for molecular channel extraction and visualization." *BMC Bioinformatics*, 16(1):1–19, 2015.

2. Raghavendra Sridharamurthy, Talha Bin Masood, Harish Doraiswamy, Siddharth Patel, Raghavan Varadarajan, and Vijay Natarajan. "Extraction of robust voids and pockets in proteins." In *Visualization in Medicine and Life Sciences III.* Lars Linsen, Hans-Christian Hege, and Bernd Hamann (Eds.) Springer-Verlag, *Mathematics and Visualization Series*, pages 329–349, 2016.

3. Talha Bin Masood, and Vijay Natarajan. "An integrated geometric and topological approach to connecting cavities in biomolecules." In *Proc. IEEE Pacific Visualization Symposium*, PacificVis '16, pages 104–111, 2016.

4. Talha Bin Masood, Hari Krishna Malladi, and Vijay Natarajan. "Facet-JFA: Faster computation of discrete Voronoi diagrams." In *Proc. Indian Conference on Computer Vision Graphics and Image Processing*, ICVGIP '14, pages 20:1–20:8. ACM, 2014.

5. Talha Bin Masood, Tathagata Ray and Vijay Natarajan. "Parallel computation of alpha complex for biomolecules." *Under review, Computational Geometry: Theory and Applications*.

Contents

List of Figures

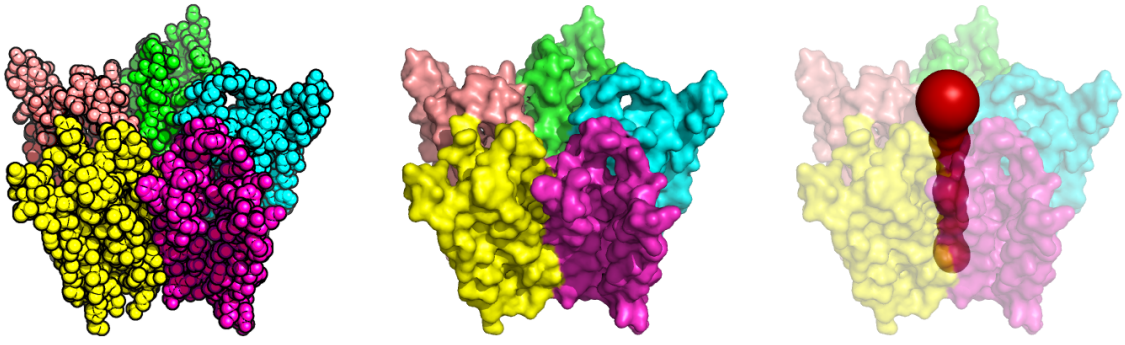
List of Tables

Chapter 1

Introduction

Fundamentally all biological processes are molecular in nature. So, it is essential to understand biomolecules and their interactions to gain better insight into living systems. Proteins are constituted of chains of small building blocks called amino acids. These chains of amino acids fold in 3D space to define structure of a protein. It is known that structure of biomolecules plays an important role in defining its function. As evident from Figure 1.1(c), biomolecular structures contain complex features such as pockets and protrusions on the surface, internal cavities and voids, channel and tunnel like structures connecting external surface to functional sites buried deep inside the molecule [60]. Analysis of these features is very important for understanding of structure-function relationships, engineering new proteins with required functional properties, or designing inhibitors for existing proteins.

As shown in Figure 1.1(a) proteins are often represented in space-fill model as a union of balls,



(a) (b) (c)

Figure 1.1: ACH receptor transmembrane protein (PDB id: 1OED). (a) The space-fill model. (b) The molecular surface. (c) The central transmembrane pore through this protein.

where each ball corresponds to an atom. This model is ideal for application of geometric and topological techniques for detailed analysis. For example, geometric algorithms have been developed for extraction of molecular surface (see Figure 1.1(b)) which is extremely important in the study of any protein. Similarly, for accurate measurement of molecular volumes, identification and characterization of empty space within a molecule, methods from computation geometry and topology are applied. This thesis is a contribution to this area of research with special focus on integrated geometric and topological methods for visual analysis of cavities and channels in biomolecules. Refer to Figure 1.2 for a 2D illustration of features of interest in a biomolecule and a brief overview of alpha complex based representation of biomolecular space [32] which we use extensively for extraction of these features in this thesis.

With increasing availability of structures of large proteins and protein complexes at atomic detail through advancements in the field of crystallography, there is a need of designing faster and more space efficient algorithms for their analysis. Another driver for the need of efficient geometric

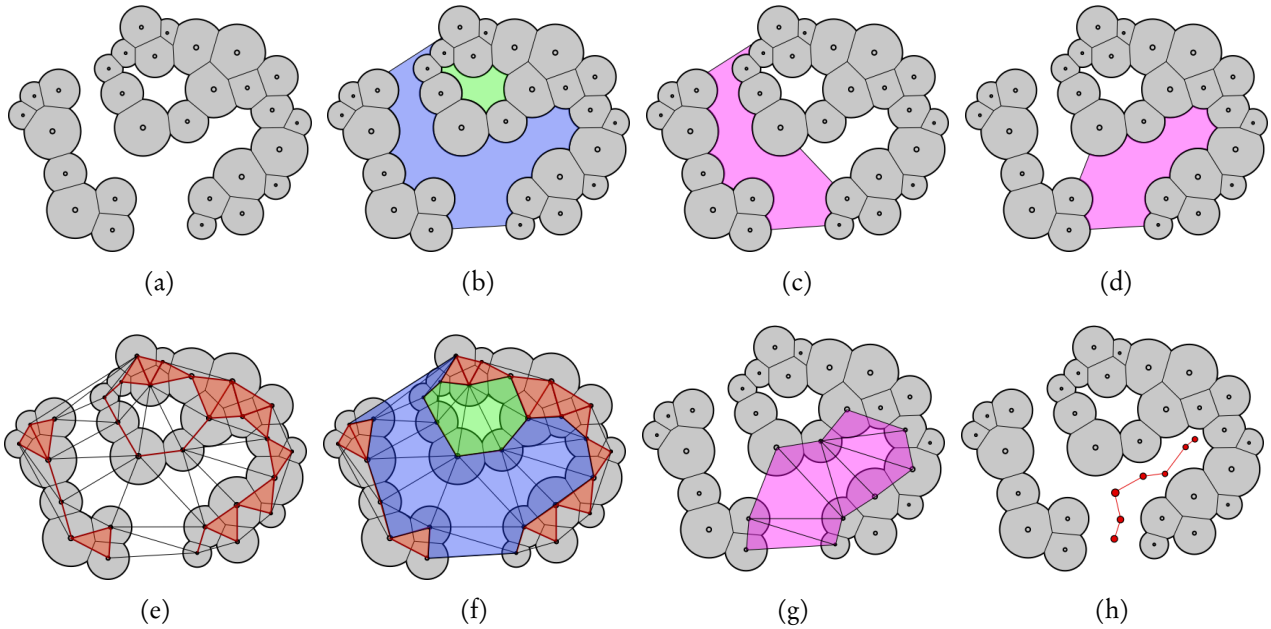


Figure 1.2: 2D illustration of features of interest in a biomolecule. (a) Set of disks representing a molecule in 2D. (b) Two cavities in the molecule. The green cavity is surrounded by atoms, is therefore called a *void*. The blue cavity here has two openings, and thus called a *pocket*. (c) A *pore* is a channel through the molecule. (d) While a *tunnel* is a channel which leads to a binding site in the molecule. (e) The weighted Delaunay triangulation for this set of disks. The alpha complex is shown in red. (f) The two cavities can be detected as maximally connected components in the complement of alpha complex. (g) Similarly, the tunnel is a set of connected triangles in the complement. (h) The dual of the set of triangles provides the path representation of the tunnel.

algorithms is the availability of larger molecular dynamics trajectories, which are essentially time varying molecular structures. Designing algorithms to address these challenges is the second major focus of this thesis.

1.1 Extraction and visualization of channels

A *channel* is a pathway through empty space within the molecule. Understanding channels, that lead to active sites or traverse the molecule, is important in the study of molecular functions such as ion, ligand, and small molecule transport. Efficient methods for extracting, storing, and analysing protein channels are required to support such studies. We develop an integrated framework that supports computation of the channels, interactive exploration of their structure, and detailed visual analysis of their properties [75]. Key contributions are summarized below:

- We describe a method for extraction of channels in biomolecules based on a representation of the molecule using the alpha complex. This is exploited to capture all geometrically feasible channels in a concise representation called *channel network* that supports querying for specific channels. The extracted channels are represented as a set of connected tetrahedra.
- Novel methods are developed to automatically identify important channels within the net-

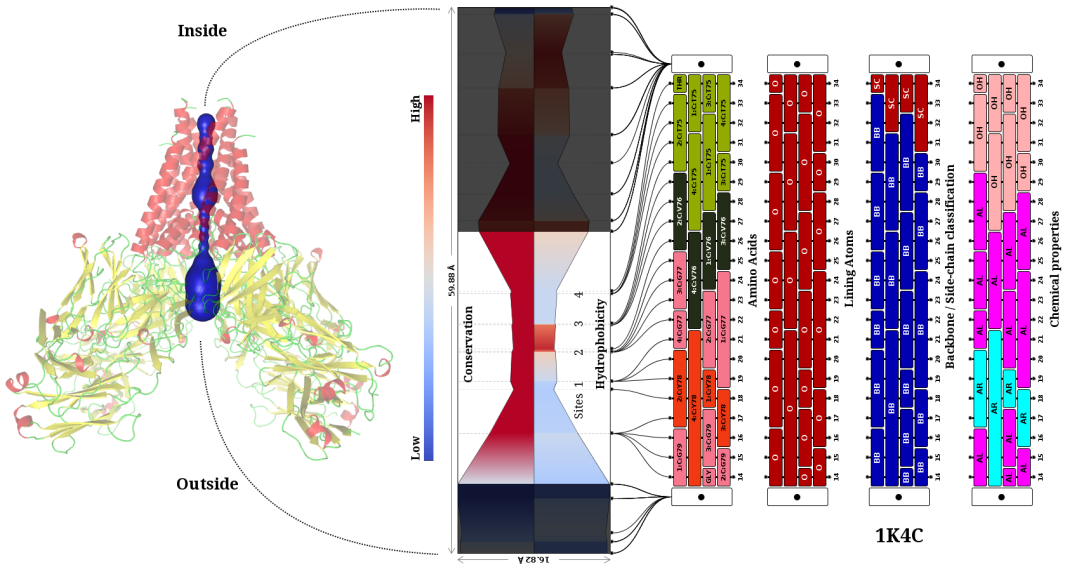


Figure 1.3: Transmembrane pore identified in PDB structure 1K4C using our channel extraction method. The 3D view of the channel is shown on the left. Conservation and hydrophobicity profiles are shown using a blue to red color map in the middle. Four different 2D box representations of the channel are shown on the right. From left to right, boxes are labelled by amino acid type, atom type, structure and chemical properties of the lining atoms.

work and rank them based on their significance.

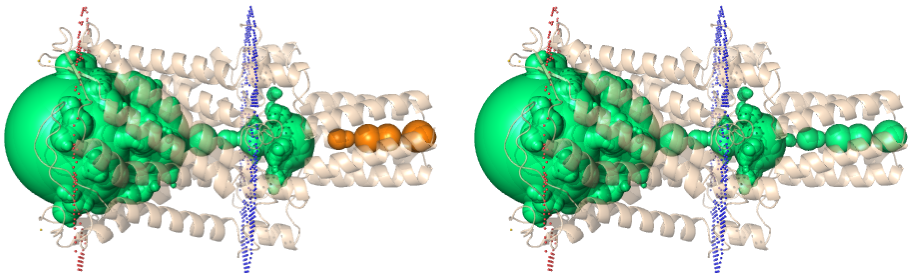
- The channel extraction method was compared with the existing software tools. The quality of the results was observed to be better than or comparable to other tools – MOLE, CAVER, MOLAXIS, and POREWALKER.
- Novel visualization methods are proposed to facilitate detailed study of the extracted channels. Figure 1.3 shows visualization of potassium channel in a transmembrane protein.
- The integrated channel extraction and visualization framework was successfully used to study multiple transmembrane pores and channels leading to active sites.
- These methods are implemented as a web-server called CHEXVIS[1] which is available for public use.

1.2 Connecting cavities in biomolecules

The tools and techniques developed for extraction of cavities in molecules are sensitive to uncertainties in atomic position and radii (Refer to Figure 1.4 for an example). We study the problem of cavity extraction in biomolecules while taking into account such uncertainties [73, 90]. We propose a simple and direct approach to address this problem, where the user examines the cavities and identifies artifacts or undesirable disconnections. The user interacts with the multiple linked views provided by the visualization and specifies a pair of cavities to be connected. Our cavity connection algorithm efficiently and automatically computes an optimal conduit between the cavities. Key contributions include:

- A simple, explicit, and flexible method for extracting cavities in biomolecules from uncertain data with guaranteed bounds on the perturbation required.
- Efficient algorithms to compute a conduit between user selected cavities that satisfies well defined optimality criteria.
- Interactive visualization of cavities in a molecule with multiple linked views that facilitates identification of disconnected cavities.
- Case studies that demonstrate the benefits of the cavity connection based method.
- These methods are implemented as a web-server called ROBUSTCAVITIES[2] which is available for public use.

[1] http://vgl.csa.iisc.ac.in/chexvis/
[2] http://vgl.csa.iisc.ac.in/robustCavities/

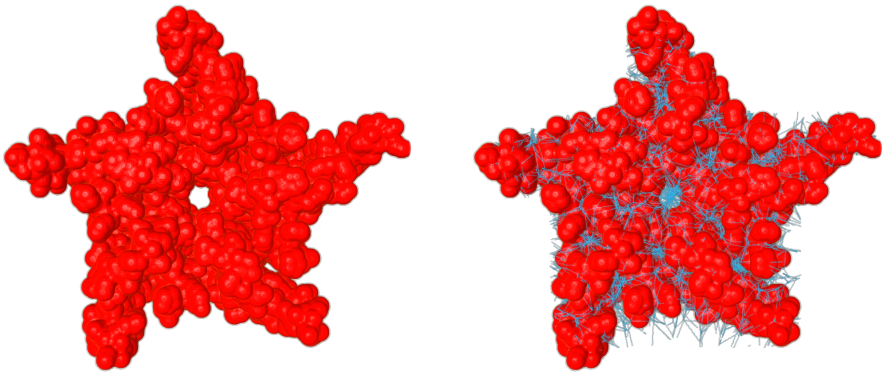


(a) (b)

Figure 1.4: (a) Biologically relevant transmembrane pore in the protein (PDB id: 2OAR) is identified as two disconnected cavities using existing methods. (b) Using our approach, we find the best path to connect the two cavities and correctly identify the biologically significant pore.

1.3 Parallel computation of discrete Voronoi diagram

Voronoi diagram, a partitioning of space based proximity to input point sites, is one of the most widely studied structures in computational geometry. In the context of structural biology, it plays a key role in the identification of channels in proteins, computation of molecular surfaces, determining depth of binding sites, etc. Voronoi diagram also finds applications in computer graphics, image processing, mesh processing, robot navigation, and for data analysis in several scientific and engineering disciplines. In most cases, Voronoi diagrams are computed for the continuous case where the space of interest is 2D or 3D Euclidean space and the input set of sites is finite. Discrete Voronoi



(a) (b)

Figure 1.5: (a) A transmembrane protein (PDB id: 1OED). (b) The channel network extracted using FACET-JFA.

diagram however requires the computation of regions on a discrete grid of finite pixels, with seed points being some of the pixels themselves. Jump Flooding can be used to generate discrete Voronoi diagrams. We introduce a variant of JFA, called FACET-JFA, wherein only the pixels that are located near the Voronoi region boundaries are processed, thus immensely reducing the total amount of work done by the algorithm [74]. The key results are summarized below:

- A novel variant of JFA, called FACET-JFA. This strategy enables both space optimization and better running times in practice. We observed upto 10x speed-up over JFA using FACET-JFA.
- The algorithm uses an intrinsic quad tree-based approach and requires only $\log n$ steps to compute the Voronoi diagram for an $n \times n$ grid of pixels.
- A GPU accelerated technique for extraction of the channel network in biomolecules in two and three dimensions which uses FACET-JFA (See Figure 1.5).
- The proposed method allows extraction of channels at real-time interactive rates and is thus suited for visual analysis of static and dynamic channel structures in Molecular Dynamics (MD) simulation trajectories.

1.4 Parallel computation of alpha complex

Alpha complex, a subset of Delaunay triangulation, has been extensively used as a tool in the study of biomolecular structures. It is crucial for the accurate computation of geometric properties of biomolecule like volume and surface area. We propose an algorithm that avoids the expensive Delaunay triangulation computation and instead directly computes the alpha complex for biomolecules. The key contributions are summarized below:

- A new characterization of the alpha complex – a set of conditions necessary and sufficient for a simplex to be a part of the alpha complex.
- A new algorithm for computing the alpha complex of a set of weighted points in $\mathbb{R}^3$. The algorithm identifies simplices of the alpha complex in decreasing order of dimension without computing the complete weighted Delaunay triangulation.
- An efficient CUDA based parallel implementation of this algorithm for biomolecular data that can compute the alpha complex for a 10 million point dataset in approximately 10 seconds.
- A proof of correctness of the algorithm and comprehensive experimental validation to demonstrate that it outperforms existing methods.

1.5 Organization

This thesis is organized as follows. Chapter 2 discusses the necessary mathematical background for representation of biomolecular structure. Chapters 3-5 constitute the first part of the thesis where we focus on tools and techniques designed to address the challenges in cavity and channel extraction from end user perspective. Specifically, in Chapter 3 we describe a tool called ChExVis designed for extraction and visualization of channels in biomolecules. Chapter 4 discusses a web server tool called RobustCavities proposed for extraction of robust cavities in uncertain data. Chapter 5 describes an integrated geometric and topological approach to connecting cavities in biomolecules to facilitate exploration of cavities in uncertain data. Chapters 6 and 7 constitute the second part of the thesis where we focus on designing efficient parallel algorithms for two geometric structures widely used in the study of biomolecules. Specifically, Chapter 6 presents a GPU based parallel algorithm for computation of discrete Voronoi diagrams, while Chapter 7 discusses a GPU based algorithm for computation of alpha complex for biomolecules. Chapter 8 concludes the thesis.

Chapter 2

Mathematical Background

Here, we briefly introduce the mathematical background required to define and represent the structure of biomolecules. We use the alpha complex based representation of biomolecular structure, which is described in detail in a book by Edelsbrunner [31]. For technical background on computational topology, the reader can refer to another book by Edelsbrunner [32]. In this chapter, we will only describe the mathematical objects relevant for this thesis in short. For details, the reader is referred to the above mentioned books.

2.1 Simplicial complex

A *d-simplex* σ is the convex hull of $d+1$ affinely independent points. A vertex, edge, triangle, and tetrahedron are d-simplices of dimension 0 to 3. A simplex τ is a *face* of σ, $\tau \leq \sigma$, if it is the convex hull of a non-empty subset of the $k+1$ points. A simplex σ is called the *coface* of τ if τ is a face of σ.

A *simplicial complex* K is a finite collection of simplices such that (a) $\sigma \in K$ and $\tau \leq \sigma$ implies

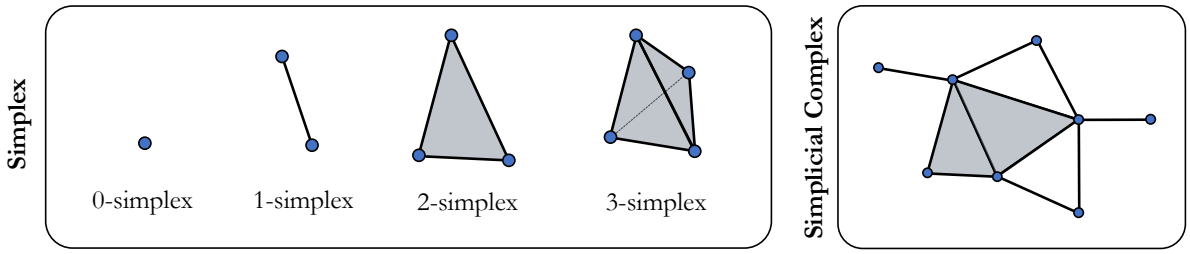


Figure 2.1: **Left:** Simplices of dimension 0 to 3, **Right:** An example of a two dimensional simplicial complex.

$\tau \in K$, and (b) $\sigma_1, \sigma_2 \in K$ implies $\sigma_1 \cap \sigma_2$ is either empty or a face of both σ_1 and σ_2. Refer to Figure 2.1 for an example. A *subcomplex* of K is a simplicial complex $L \subseteq K$.

2.2 Biomolecule representation

Protein molecules are often modelled as union of balls. The molecule B is defined as the set $\{b_i = (p_i, r_i)\}$. Here b_i denotes a constituent atom of B modelled as a ball with center at p_i and radius r_i.

2.2.1 Voronoi diagram and Delaunay triangulation

Let $S = \{s_1, s_2, \cdots, s_k\}$ be a finite set of points (also called *sites*) in $\mathbb{R}^d$ in general positions. The *Voronoi cell* of a site $s_i \in S$ is the set of points in $\mathbb{R}^d$ whose Euclidean distance to s_i is smaller than

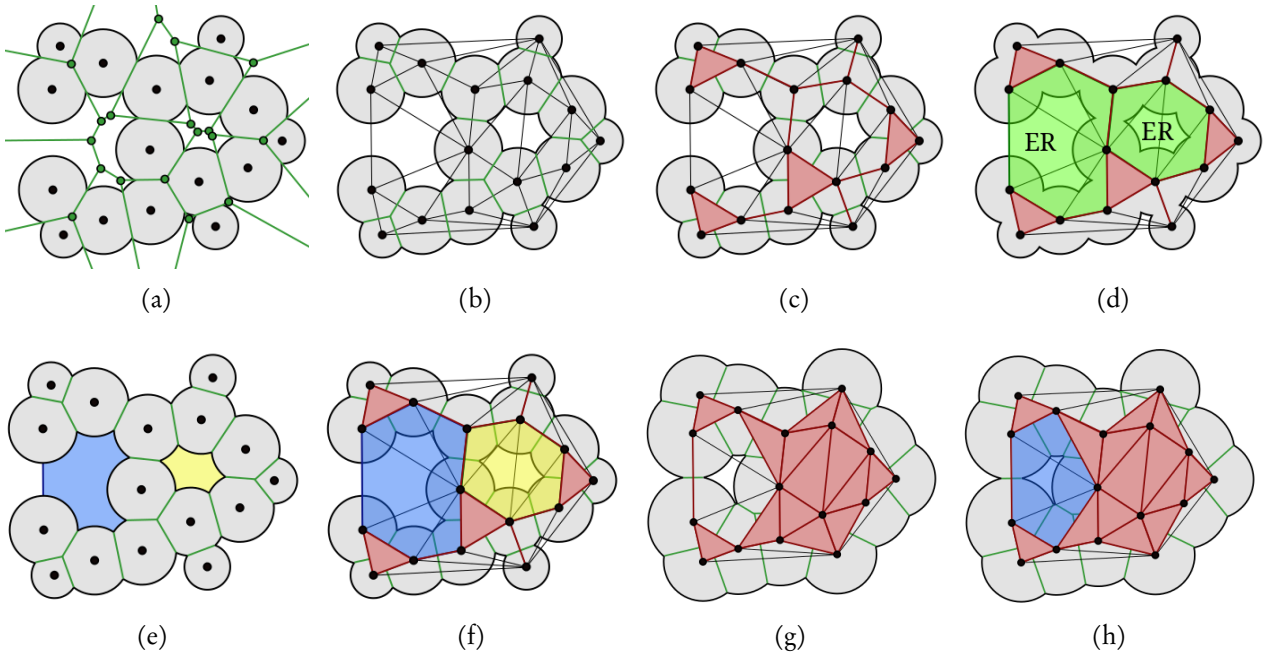


Figure 2.2: (a) Voronoi diagram of a weighted point set in $\mathbb{R}^2$, Voronoi edges are in green. This is also called *Power diagram*. (b) The weighted Delaunay complex is the dual of the power diagram. (c) The α-complex K_α for $\alpha = 0$ is shown in red. This is the dual of the intersection of power diagram and union of balls. K_α forms the occupied region (OR) of the molecule. (d) The empty region (ER) in green. This region is defined by Delaunay flow. The green triangles do not belong to OR and have flow towards a triangle within the molecule. The set of simplices in OR and ER form the molecular region (MR). (e) Void and pocket in a collection of 2D balls. Void is shown in yellow and pocket is shown in blue. (f) ER consists of two maximally connected components, called cavities, shown in blue and yellow. (g) The alpha complex shown for some $\alpha > 0$ where the void has been filled up and original pocket has become a void. (h) The new void is highlighted using blue, Delaunay edges (in black) and alpha complex (in red) are also shown to provide context.

or equal to any other point in S. The collection of Voronoi cells of all the sites in S partitions $\mathbb{R}^d$. In 3D, the Voronoi cells are 3D convex polyhedrons. The boundary of Voronoi cells are convex polygons and are called the *Voronoi faces*. The points on Voronoi faces are equidistant from two input sites. The boundaries of Voronoi faces are line segments, and referred to as the *Voronoi edges*. The points on Voronoi edges are equidistant from three input sites. The Voronoi edges meet at points called *Voronoi vertices*. The Voronoi vertices are equidistant from four input sites. This geometric structure of collection of Voronoi cells, along with Voronoi faces, Voronoi edges and Voronoi vertices is referred to as the *Voronoi diagram*.

The *Delaunay triangulation* D of S is the dual of the Voronoi diagram and partitions the convex hull of S. The dual mapping is done such that k-dimensional Voronoi element is mapped to $(d-k)$-dimensional simplex in Delaunay triangulation. So, the dual of a Voronoi cell is a point, the input site corresponding to the Voronoi cell. The dual of a Voronoi face is a *Delaunay edge* corresponding to the two input sites whose Voronoi cells share this Voronoi face. Similarly, the dual of Voronoi edge is *Delaunay triangle* incident on three inputs sites, intersection of whose Voronoi cells create this Voronoi edge. Lastly, the dual of Voronoi vertex is a *Delaunay tetrahedron* incident of four input sites whose Voronoi cells meet at this Voronoi vertex. Clearly, the Delaunay triangulation is a simplicial complex and is referred to as the *Delaunay complex*. One important property which tetrahedra in Delaunay triangulation satisfy is called the *empty circumsphere property*, which states that circumsphere of a Delaunay tetrahedron should not contain any other input site. This property has been exploited in computation of 2D Delaunay triangulations where a given triangulation is modified incrementally until all the triangles satisfy the empty circumcircle property.

The definitions of Voronoi diagram and Delaunay triangulation can be extended to the case where the input sites are replaced by balls. These are called *weighted Voronoi diagram* and *weighted Delaunay triangulation*, respectively. The distance between an input ball b_i, with center p_i and radius r_i, and a point $q \in \mathbb{R}^d$ is given by the *power distance* $\pi(b_i, q) = \|q - p_i\|^2 - r_i^2$. Similar to the case of Voronoi diagram, $\mathbb{R}^d$ can be partitioned based on proximity to input balls in terms of the power distance. The Voronoi cell of an input ball $b_i \in B$ is the set of points in $\mathbb{R}^d$ whose power distance to b_i is smaller than or equal to any other point in B. The collection of Voronoi cells of all the balls in B is called the *weighted Voronoi diagram* or *Power diagram*. The *weighted Delaunay triangulation* D of B is the dual of the weighted Voronoi diagram. The weighted Voronoi diagram and the weighted Delaunay triangulation are shown in Figures 2.2(a) and 2.2(b), respectively. The reader can refer to the book by Aurenhammer [4] for a comprehensive account of Voronoi diagrams and Delaunay triangulation, and their weighted avatars.

2.2.2 Alpha complex

The *alpha complex* is defined as a subcomplex of the weighted Delaunay complex based on a growth model on the balls that is consistent with the power distance. The growth parameter α corresponds to a radius $\sqrt{r_i^2 + \alpha}$ for a ball b_i centered at p_i with radius r_i. Positive values of α correspond to growing the balls and negative values correspond to shrinking the balls. The weight w_i of the ball b_i is defined as $w_i = r_i^2 + \alpha$. The weight increases or decreases by α between $-\infty$ and ∞. Note that $\alpha = 0$ corresponds to no growth. The parameter α can be varied from $-\infty$ to ∞ to obtain a filtration of simplices belonging to the weighted Delaunay complex. A sequence of α-complexes $(\emptyset = K^0 \subset K^1 \subset \cdots \subset K^m = D)$ containing progressively more triangles and tetrahedra is obtained as α is increased from $-\infty$ to ∞. Figure 2.2(c) shows the α-complex at $\alpha = 0$ as a subset of D highlighted in red. Figure 2.2(g) shows alpha complex at for some $\alpha > 0$. Clearly, more simplices are part of the alpha complex at higher values of α; the input disks are also enlarged according to α. We use the notation K_α to denote the alpha complex of B at the value α.

As α varies from $-\infty$ to ∞, topological features such as tunnels and voids appear in and disappear from the alpha complex. The importance of a topological feature is captured by the notion of *topological persistence* [40], which is equal to the length of interval of α for which the feature was present in the alpha complex.

2.2.3 Cavities

The following characterization of cavities is motivated by the work on identification of protein cavities by Liang *et al.* [64, 65, 66]. We present those ideas here in consolidated form and introduce our own notations in order to present clear definition of cavities using alpha complex.

For a given molecule represented as a set of balls, let D be the weighted Delaunay triangulation and $K_\alpha \subseteq D$ be the α-complex for value α. The *Delaunay flow* over D is defined as the collection of flows between adjacent tetrahedra in D going from the tetrahedron from smaller circumsphere to the larger one. Let I_{tet} denote the set of tetrahedra in D whose Delaunay flow terminate within D and I denote I_{tet} together with the corresponding faces. For a given α value, we define *molecular region* MR as $K_\alpha \cup I$. The simplices in MR can be classified into two groups based on whether they belong to the α-complex or not. The simplices in $OR = K_\alpha$ constitute the *occupied region* in the molecule, while the remaining simplices $ER = MR - OR$ capture the *empty region* in the molecule. Refer to Figure 2.2(d) for an illustration of ER, OR and MR. In the figure, the green region is ER while red region is OR. The union of ER and OR is the molecular region, MR. The *cavities* are defined as maximally connected subregions in ER. Let the set of all cavities be $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, such that $ER = C_1 \cup C_2 \cup \cdots \cup C_k$ and $C_i \cap C_j = \emptyset$. The tetrahedron $t_i \in C_i$

with highest α value is selected as the *representative tetrahedron* of C_i. Refer to Figures 2.2(e) and 2.2(f) for an illustration of cavities.

The cavities which are completely surrounded by atoms of the molecule are called *voids*, while the cavities which have openings to region outside the molecule are called *pockets*. The openings of these pockets are sometimes referred to as *mouths*. The pockets can be detected as voids in alpha complex for some higher values of α. For example, the pocket shown in blue in Figure 2.2(f) becomes a void at higher value of α as shown in Figure 2.2(h).

Chapter 3

ChExVis: A Tool for Molecular Channel Extraction and Visualization

Protein channels are crucial for transport of ions, ligands, solvents and other macromolecules. Such channels occur in diverse systems such as enzymes where they play a role in navigating the ligand to a buried active site, or in channelling an intermediate through multiple entry and exit pathways, or in membrane proteins that are involved in the transport of small molecules, ions *etc*. The selectivity of the channels in permitting access to specific types of molecules and the micro-environment that it provides is crucial to the nature of the molecule that it transports [101]. Indeed, it is important to identify and study channels since mutations in residues lining the channel have resulted in channel dysfunctions. Such channelopathies have been associated with defective insulin secretion, diseases such as cystic fibrosis, epilepsy and kidney stone disease [53].

Geometrically, a *channel* is a pathway through the empty space within a molecule that connects an internal point and the molecular exterior [87]. A channel that passes through the molecule and connects two exterior points is called a *pore*. Other terms like *tunnel* and *molecular path* have also been used to refer to channels. However, we will consistently use the term channel to refer to both simple channels and pores. In this chapter, we study the problem of efficient computation and effective visual exploration of channels in biomolecules. There is a need for an integrated framework that supports computation of the channels, interactive exploration of their structure, and detailed visual analysis of their properties. Although there exist tools that partly address this need, they either do not guarantee a robust computation of channels or they are found lacking in providing sufficient support for interactive visualization of channels and their properties. We aim to address these shortcomings, and develop a tool that uses sound mathematical theory for extraction of channels and also supports wide variety of intuitive and useful visualizations of channels and their properties.

Related work

In recent years, numerous computational methods have been developed for detection and classification of empty spaces in proteins. Early techniques focused on finding cavities and pockets in molecules. These included grid-based approaches such as POCKET [63], LIGSITE [49] and VICE [96]. To overcome the inaccuracy of grid based methods, geometric and topological techniques were exploited to find cavities, more accurately, in software like CASTp [29], CAVER [20] and ProShape [56].

The problem of channel extraction[1] was first addressed in HOLE [88]. The proposed solution involved splitting the molecule into slices along a user-specified vector and determining the largest empty sphere within each slice using simulated annealing. Similar approaches were used in other tools as well, most notably POREWALKER [79]. The idea of approximating the molecular space as a grid and determining channels by processing grid voxels has also been exploited in tools such as dxTuber [82], HOLLOW [51], 3V [98] and CHUNNEL [23]. Although this approach is computationally efficient, the accuracy depends on the grid resolution. Voronoi diagram based techniques avoid the need to choose approximate grid resolutions by directly representing balls and the space they occupy. However a key assumption is that the ion or molecule that traverses the channel may be represented by a ball. This approach is followed in MOLE [87, 80], MOLAXIS [100], CAVER [20, 57] and state of the art techniques developed by Lindow *et al.* [68, 69] and Kim *et al.* [55]. MOLE uses pruned Voronoi diagram of atom centres for extracting channels. MOLAXIS and CAVER support differing atomic radii by approximating large atoms as a union of small balls with uniform radii. Lindow *et al.* compute the Voronoi diagram of spheres to further improve the geometric accuracy of channel centerlines. Our proposed channel extraction technique falls in the category of Voronoi diagram based methods. Different from the above, we use the alpha complex, which is based on the power diagram, to compute channels in biomolecules. The channels computed using this approach are guaranteed to be feasible. Various channel extraction techniques are studied and compared in a recent detailed review [12].

Many of the above-mentioned methods and software tools do not facilitate study of physico-chemical properties of the extracted channels, and focus only on computing geometric properties of the channel. After a recent update, MOLE [87] supports computation of some physico-chemical properties. However, there is a lot of scope for improvement in terms of how these properties are visualized and presented to the user.

[1] We use the term *channel extraction* because we view the problem as a special case of the *feature extraction* problem. The features of interest are channels and the data is a biomolecule represented in PDB format.

Contributions

We describe a method for channel extraction, based on the alpha complex, that ensures extraction of geometrically feasible channels. Our proposed alpha complex-based representation of channels allows the storage of both the volumetric region occupied by the channel and its centreline in a unified manner. We demonstrate the usefulness of this approach *via* multiple case studies describing the automatic extraction and ranking of transmembrane pores. To gain a deeper functional understanding of extracted channels, we propose the computation of channel profiles. The second contribution is the development of simple but information rich representations for effective visualizations of the channel and its profiles. The methods presented in this chapter are implemented within a software tool called ChExVis, which is available as a web-server. Finally, ChExVis is compared with other channel extraction tools in terms of the supported features and evaluated based on the ability to identify known channels in a set of transmembrane proteins.

3.1 Channel extraction

In this section, we describe a method of channel extraction based on alpha complex. The reader should refer to Chapter 2 for the necessary technical background. Figure 3.1 provides a brief refresher. As shown in Figure 3.1(c), the alpha complex and its complement partition the weighted Delaunay triangulation. For the case of $\alpha = 0$, the alpha complex represents the region covered by the atoms in the molecule. The complement of alpha complex, consisting of the remaining tetrahedra, triangles and edges of the weighted Delaunay triangulation, represents the empty space within the molecule.

The method proceeds as follows. First, a network of all geometrically feasible channels is constructed using the complement of alpha complex. Then, significant channels leading to active sites and important pores are identified within this network, depending on user-specified input. For identifying important pores, important end points are determined based on topological persistence [40]. A significant pore is reported for each pair of important end points. In the case of transmembrane proteins, a set of pores that traverse the membrane are identified and ranked in order of significance. Each extracted channel has a tetrahedral representation which accurately captures the volume occupied by the channel. The tetrahedral representation is exploited to efficiently compute channel profiles for every channel identified above. These profiles can be interactively visualized along with various 3D representations of the channel in the context of the biomolecular structure.

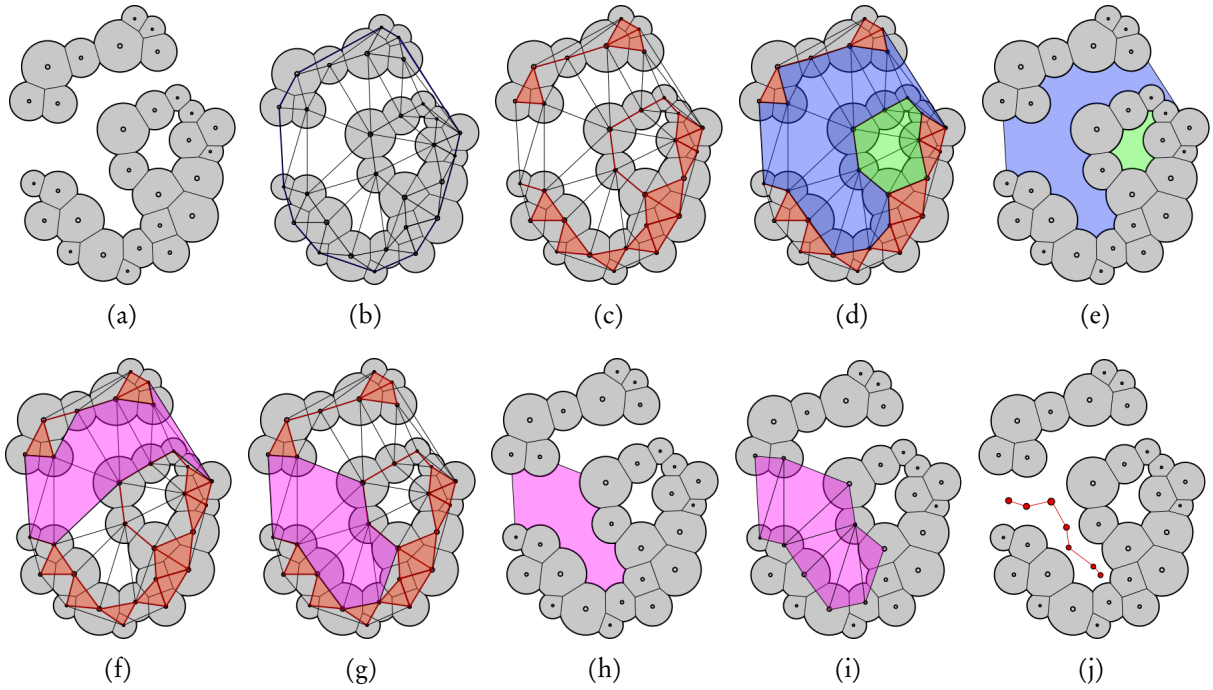


Figure 3.1: 2D illustration of an alpha complex based representation of a molecule and the empty space within. (a) Union of disks (balls in 3D represent atoms) where the contribution from each disk is equal to its intersection with the corresponding Voronoi cell. (b) The weighted Delaunay triangulation of the disks and the convex hull (bold). (c) Alpha complex at $\alpha = 0$, shown in red, is a subcomplex of the weighted Delaunay triangulation. (d) A cavity is a connected component of the complement of the alpha complex. A cavity with atleast one opening is a pocket (blue), while buried cavities are referred to as voids (green). (e) The empty space represented by the cavity triangles. (f) A channel is a simply connected subset of simplices of a pocket each of whose triangles has at most two neighbours and at least one boundary edge is a mouth edge. Here a pore (pink), a channel with two openings, is shown represented as a subset of the complement of the alpha complex. (g) A channel from the boundary to an interior point. (h) Underlying empty space of the channel. (i) Simplices of the complement of the alpha complex that represent the channel. (j) A path representation of the channel in which nodes are located at the centers of the orthogonal circle corresponding to each triangle and arcs connect nodes that correspond to neighbouring triangles.

3.1.1 Channel network

All channels lie within the empty space of the molecule by definition. Therefore, we restrict our focus to those tetrahedra, triangles, and edges of the weighted Delaunay triangulation that do not belong to the alpha complex. We construct the dual graph of the complement to obtain a *channel network* in the molecule. The channel network is pruned by restricting to nodes that are accessible from the exterior because the empty space within voids is not accessible. The channel network is a subset of the power diagram of the set of atoms. A triangle in the empty space that lies at the

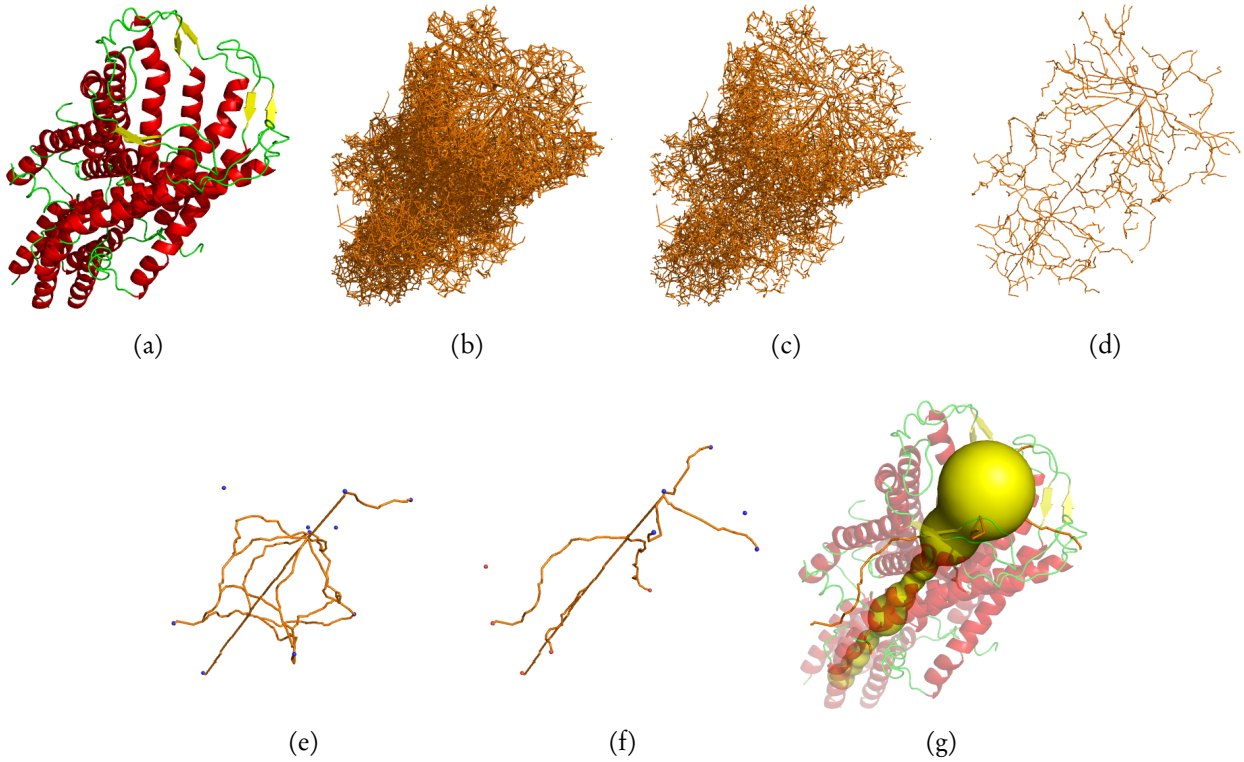


Figure 3.2: Channel extraction in a transmembrane protein. (a) Mechanosensitive Channel of Large Conductance (MscL, PDB id 2OAR). (b) The path network. (c) Widest path tree. (d) Pruned widest path tree. (e) Pores between top 10 boundary nodes in the network. (f) Transmembrane pores between top 5 interior and 5 exterior nodes. (g) Top ranked transmembrane pore shown using the skin surface (yellow). Other pores are shown for context.

interface of the molecular exterior and interior is called a *mouth* triangle, and represents an entry point into the molecule. The nodes in the channel network corresponding to tetrahedra incident on mouth triangles are called *boundary nodes*. Figure 3.2(b) shows the channel network for the transmembrane protein 2OAR. Algorithm 1 computes an annotated channel network, given the set of atoms in the molecule. The worst case running time complexity of this algorithm is defined by running time complexity of weighted Delaunay triangulation which is known to be $O(n^2 \log n)$ for 3D input where n is the number of atoms. The other steps of the algorithm are linear in the number of output simplices.

We also extract the *widest path tree* in the channel network by formulating it as a maximum spanning tree computation. We will discuss this tree in more detail later in Chapter 5. Intuitively, the ligands would prefer widest paths as geometrically they are paths of least resistance and least obstruction, hence this tree provides a good overview of the channel structures in the molecule, see Figure 3.2(c). The widest path tree can be further pruned so that it is a collection of paths between

Algorithm 1 Construct Channel Network

Input: S: Set of atoms.
Output: $CN = (N, E)$: Annotated channel network.
Output: $B \subset N$: Boundary nodes.

1: D := Weighted Delaunay triangulation of S.
2: K_α := Alpha Complex at $\alpha = 0$.
3: $CN = (N, E)$:= Dual graph of the tetrahedra and triangles in the set $D - K_\alpha$.
4: Prune CN by restricting it to a subset whose nodes lie inside the pockets.
5: Mark a node as a boundary node and insert into B if it is adjacent to a pocket mouth.
6: For each node in N, store the four atoms centered at vertices of the corresponding tetrahedron and the power distance.
7: For each edge in E, compute and store the length of edge and the minimum power distance along the edge.
8: **return** CN, B

important nodes as shown in Figure 3.2(d). Although there is no guarantee that all biologically functional channels will be successfully identified in the pruned tree, we observed this to be true in most cases.

3.1.2 Significant channels

There exists multiple channels within the channel network between a given pair of nodes. We aim to identify a small set of potentially significant channels, one or more of which may correspond to biologically significant channel in the molecule. Short and wide channels are considered significant. We quantify significance by assigning a cost to each edge in the channel network equal to the ratio of the edge length to the minimum power distance along the edge. The power distance is an indicator of the local width of the channel. Next, given a set of endpoints, we compute the shortest paths between all pairs of endpoints using Dijkstra's algorithm on the channel network. Dijkstra's algorithm is executed multiple times on the network to obtain a set of important channels instead of a single most significant channel. After each iteration, weights on edges belonging to the detected channel are made high enough to ensure that subsequent iterations report edge disjoint channels. It should be noted that the naive strategy of executing Dijkstra's algorithm only once and choosing top channels, usually yields a set of very similar channels. In the discussion below, we distinguish between the case when one of the end points is an active site and buried within the molecule and the case when both endpoints lie on the boundary of the molecule.

For a molecule containing n atoms, the number of tetrahedra and triangles in D is at most $O(n^2)$. Hence, the size of CN is also bounded by $O(n^2)$. Let the number of endpoints be k. The worst case running time complexity of computing significant channels between all pairs of endpoints is

therefore $O(k^2 \cdot n^2 \log n)$. It should be noted that for inputs with well distributed points as in the case of biomolecules, the size of D is linear in n resulting in much faster running times in practice.

3.1.3 Channels to active sites

Enzymes contain buried active sites and it is often useful to compute and visualize channels leading to such sites. We accomplish this by first automatically determining a node from the channel network to represent the active site. Dijkstra's shortest path algorithm is employed to compute the channel to the boundary. We use the Catalytic Site Atlas [45] and HETATM records of significant ligands in the PDB file to determine important sites within the given molecule. Both methods provide a set of atomic locations ($A = \{p_1, p_2, \cdots, p_n\}$) from the active site. A representative node may be determined by computing the node closest to the centroid of the atoms that constitute the active site [87]. However, this approach may not work for large ligands where the centroid is typically far removed from the deepest point in the active site that interacts with the ligand. We assign a *depth* value to each node in the channel network via an iterative wave-front propagation algorithm that begins at boundary nodes and proceeds towards the interior. For each atom in the set A, we determine the closest node and hence construct the set $N(A)$. We select the node $n \in N(A)$ with the highest depth value as the representative node for the active site.

3.1.4 Extraction of pores

A channel both of whose endpoints are incident on mouth triangles is called a pore. Technically, any path between two boundary nodes of the channel network is a pore.

Important pores. Computing significant channels between all pairs of boundary nodes to determine pores is a costly operation because the number of boundary nodes is usually large. We select a representative node from each mouth. Specifically, we locate the mouth triangle with the highest persistence [37] and choose the corresponding node as the representative. This mouth triangle roughly corresponds to the widest opening of the mouth. Our choice of the value of α is conservative. We determine $\alpha_{max} \in [0, \infty)$ at which the number of mouths attains a maximum value. The mouths are extracted at α_{max} and the triangle with the maximum persistence is chosen as the representative. The above procedure reduces the list of boundary nodes to a manageable size. Let B_{imp} denote the reduced list of nodes. This list is sorted in decreasing order of the persistence value of the tetrahedron corresponding to each node. Next, the top k boundary nodes are chosen from the list, where k is a user-defined parameter. Given k, channels between all possible pairs of the k nodes are automatically extracted as important pores in the molecule. Figure 3.2(e) shows the set of pores extracted in 2OAR for $k = 10$.

Transmembrane pores. Transmembrane pores are of special interest because of their functional importance. We use the OPM database [71] to classify nodes in B_{imp} into three sets *viz.* B_{in}, B_{out}, and B_{mem} corresponding to the nodes that lie inside, outside, or within the membrane, respectively. When OPM data is not available, the TMHMM utility [58] is used to classify the nodes. The pores are extracted such that one endpoint lies in B_{in} while the other lies in B_{out}. Figure 3.2(f) shows the set of transmembrane pores. Let TM be the set of transmembrane pores extracted using this approach. We rank the pores in TM based on three criteria – length, bottleneck width, and straightness. Straight pores suggest highly regular and symmetric cavity structure around the channel centreline [79]. Straight pores are therefore preferable over winding pores in transmembrane proteins. The score $f(x)$ assigned to a transmembrane pore $x \in TM$ is defined as follows:

$$f(x) = \frac{1}{3}\left(\frac{|x|}{\max_{y \in TM}|y|} + \frac{bn(x)}{\max_{y \in TM} bn(y)} + s(x)\right),$$

where $|x|$ is the length of the pore x, $bn(x)$ is the bottleneck width of the pore, and $s(x)$ is a measure of how straight the pore is. The straightness is computed by calculating the deviation of path nodes from the best fit line. Let P denote the ordered list of uniformly distributed sample points along the path representation of the channel. The straightness term is given by the following expression:

$$s(P) = \frac{\sum_{d=1}^{|P|/2}\sum_{i=1}^{|P|-2d} d \times \cos\angle(P[i], P[i+d], P[i+2d])}{\sum_{d=1}^{|P|/2}\sum_{i=1}^{|P|-2d} d}$$

The average curvature is computed at different scales from the set P. Curvature is approximated by the cosine of the angle formed by three equally separated sample points. The distance is varied to capture the straightness of the channel at different scales. The pore with the highest score f is reported as the best transmembrane pore. Figure 3.2(g) shows the best transmembrane channel detected in 2OAR. It should be noted that correctness of identified transmembrane pores is dependent on the accuracy of the information provided by the OPM database and TMHMM utility regarding the orientation and placement of protein with respect to the membrane.

3.1.5 Advantages of using alpha complex for channel extraction

Compared to earlier Voronoi based approaches [80, 68], our proposed alpha complex based approach has the following advantages:

- The detected channels are guaranteed to be geometrically feasible. This follows from the definition of alpha complex. Delaunay triangulation based-approach, as used in MOLE [80], can also provide tetrahedral representation for channels, but these channels are not guaranteed

to be geometrically feasible as reported earlier by Chovancova *et al.* [20].

- The tetrahedral representation of the volume occupied by the channel enables accurate computation of volumes and surface areas [72].
- The tetrahedral representation may be used for other computations such as those requiring finite element analysis.
- The tetrahedral representation is also used for computing channel profiles and hence plays a crucial role in our channel visualizations.
- Euclidean Voronoi diagram of spheres allows extraction of geometrically accurate channels [68, 55], but a channel may not necessarily be represented as a set of tetrahedra using this approach.

3.2 Channel visualization

3.2.1 Channel profiles

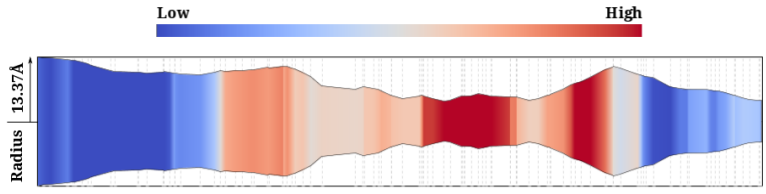
We propose the computation and visualization of *channel profiles*, 1D real-valued functions defined on the channel, with the aim of facilitating a quantitative analysis of the computed channels. Let $P \subset \mathbb{R}^3$ denote the centreline of a channel. A channel profile is a real-valued function defined on P. Let p_0 be one end-point of the channel. The centreline can be parametrized using the geodesic distance from p_0. So, the channel profiles are, indeed, real-valued functions defined on an interval $[0, d]$, where d is the length of the channel.

Radius profile. We compute the variation of the square root of power distance along the channel. We call this the radius profile because it is the radius of the orthogonal sphere to the three closest atoms. This profile provides a good estimate of the width of the channel, and is therefore useful for gaining information about the potential bottlenecks along the channel.

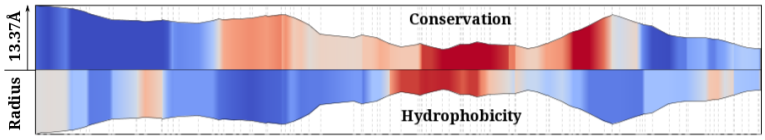
Electrostatic potential profile. Electrostatic potential is computed for the whole molecule using the finite element method APBS [6]. The computed electrostatic field is available as a sample over a grid. We use tri-linear interpolation to determine the electrostatic potential values at points along the centerline. The use of default parameters for electrostatic potential computation and approximation along centerline may lead to a situation where this profile may provide a false impression of the charge distribution inside the channel. Therefore, the electro-static potential should be used for analysis with care.

Physico-chemical profiles. We compute the average of different physico-chemical quantities over the four atoms of each tetrahedron in the channel. Important physico-chemical profiles include hydophobicity [61], charge, bulkiness, and secondary structure related profiles. We use ProtScale [46], which assigns a scalar value for each amino acid to represent a physico-chemical quantity.

Conservation profile. Conservation captures the tendency of an amino acid residue to remain unchanged over long periods of evolution. Conserved residues may be of functional significance. We use the ConSurf server [3], which uses multiple sequence alignment over a large set of sequences, to obtain conservation score for all amino acid residues in the protein. The conservation score at each node is computed as an average of the conservation scores of the four amino acids incident on the corresponding tetrahedron.



(a)



(b)

Figure 3.3: Channel profile visualization of the transmembrane pore in 2BG9. (a) Conservation profile shown as a color map over the radius profile, which in turn is shown as a symmetric graph plot. (b) Hydrophobicity and conservation profiles shown in a split visualization over the radius profile.

3.2.2 Profile visualization

Channel profiles are visualized as a simple XY-plot that shows the variation of the property along the channel. To capture the correlation of the radius profile with a second channel profile, we compute a 2D projection of the channel using the radius profile. The second profile is shown using color-mapping[1] within the resulting region, see Figure 3.3(a). This visualization may also be extended to display the correlation between the radius profile and two different profiles. The 2D projection is split into two, and each region displays a different channel profile using color-mapping as shown in Figure 3.3(b).

[1]We use blue to red diverging color map, where blue corresponds to low values while red represents high values.

3.2.3 Channel visualization in 2D and 3D

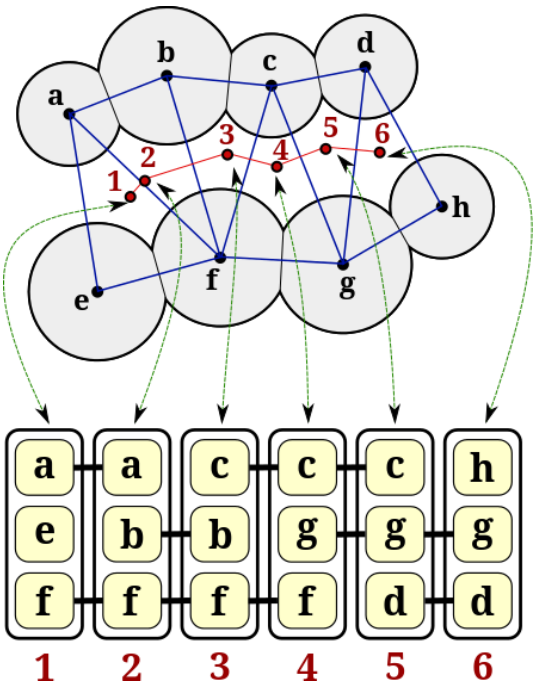


Figure 3.4: Illustration of 2D representation of a channel in a synthetic 2D example. **Top:** The molecule is shown as a set of disks (grey). The channel is shown as a path (red) and as a set of triangles (blue). In 2D, each node in the path corresponds to a triangle, while a path edge corresponds to an edge in the triangulation. Each disk is given an alphabetic label, while the nodes of the path are given numeric labels. **Bottom:** The proposed 2D visual metaphor of the channel is shown. Each vertical box denotes a node in the path (and thus denotes a triangle within the channel). The three small boxes within the node box denote the disks incident on the triangle corresponding to the particular node box. Also, consecutive nodes boxes are connected by two edges. These edges denote the edge shared by the consecutive triangles in the channel. For example, node boxes 3 and 4 denote triangles cbf and cgf, respectively. Their shared edge is cf, so the atom boxes corresponding to discs c and f are connected by edges. This representation naturally extends to the 3D case, with the only modification that each node box will denote a tetrahedron and will thus contain four atom boxes. Two consecutive node boxes will be connected by three edges denoting the common triangle shared by the adjacent tetrahedra.

We design a novel visualization metaphor for channels based on tetrahedral representation of channels. A channel is visualized using the 2D projection. We use the correspondence between the channel network nodes and the weighted Delaunay triangulation tetrahedra, and between the channel network edges and the weighted Delaunay triangulation triangles to effectively visualize different properties along the channel. Each node is displayed as a *node box* containing four smaller *atom boxes*, which correspond to the four atoms at the vertices of the tetrahedron. The channel is

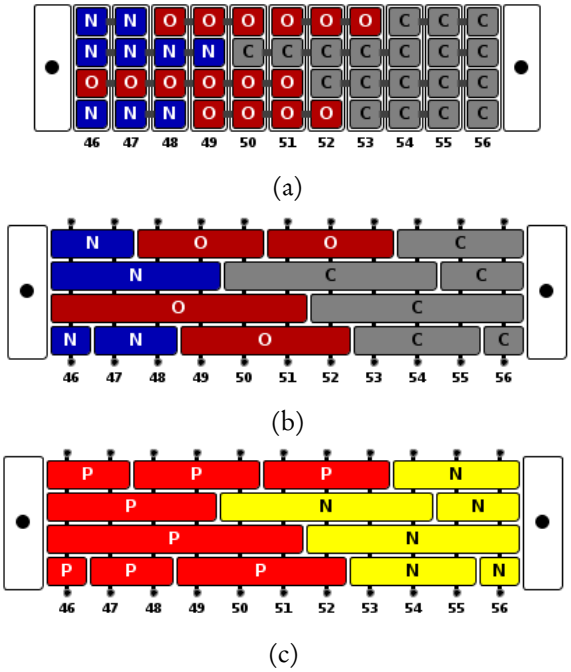


Figure 3.5: 2D representation of a channel. (a) 2D box representation of the atoms lining a portion of the channel in 2BG9. (b) and (c) Boxes are merged and coloured by atom type and polarity of residues. The numbers below the boxes are tetrahedron indices, which makes it clear that this visualization corresponds to a subset (tetrahedra 46-56) of the complete channel.

shown as a sequence of node boxes. Adjacent node boxes are connected by three edges that correspond to the triangle shared by the two adjacent tetrahedra. Figure 3.4 illustrates this representation for a 2D example, while Figure 3.5 demonstrates this visual representation in the 3D case for a part of the transmembrane pore in 2BG9. Consecutive atom boxes are merged into one if they correspond to the same atom (Figures 3.5(b) and 3.5(c)). The power of this representation is realized when we treat each atom box as a place holder. Each box is given a label and a color. Appropriate choices of a colormap and labels for the boxes based on the physico-chemical properties results in a concise and effective visualization of the channel. This representation can be viewed along with profile visualization to obtain richer information about the channel. This visualization metaphor can be used for representing any channel represented as a set of tetrahedra. In particular, this visualization may also be used by tools like MOLE to show variation of physico-chemical properties across a channel.

The channel is also visualized in 3D within the context of the molecule. We support visualizing channels as tetrahedra, union of balls, or as a surface of the union of balls. The union of balls representation of the channel is obtained by computing the orthogonal sphere for each channel tetrahedron. The skin surface is a smooth surface that wraps around the given set of spheres [18]. We compute the skin surface for the union of balls representation of the channel. In addition to

displaying the channel, we optionally show atoms or amino acid residues lining the channel. This may help the user to further inspect the physico-chemical properties of the channel. We also provide pymol scripts to facilitate download and visualization of the channel properties. The combined view of the channel and residue is pointing into the channel lumen coupled to their physico-chemical and biochemical properties and conservation in related proteins would be useful in assessing mutability of channel residues and their mapping on to the protein 3D structure.

3.3 ChExVis

The channel extraction and visualization methods presented in this chapter are implemented into a software tool called ChExVis. A web-server[1] has been implemented and made freely available. The web-server is comparable to existing state-of-the-art channel extraction software in terms of functionality.

3.3.1 Web server features

The ChExVis web-server supports submission of jobs by specifying the PDB ID or uploading a PDB file. Biological assemblies of proteins can also be uploaded. While the default values for the parameters often produce required results, the user may optionally tune multiple parameters. Further, the server also computes transmembrane pores similar to existing tools like PoreWalker that exclusively detect such channels. The unique and novel 2D representation of the channel enables a detailed analysis of its physico-chemical properties at a level of detail not supported by other existing software. The extracted channels and their properties are displayed within the browser using Java/JSMol based applets. PyMOL [28] scripts for individual channels are also generated and may be downloaded for detailed off-line study. See Figure 3.6 for typical output of the web-server. With default parameters, for typical input PDB files containing upto 20,000 atoms, the web-server takes around two minutes for computation of channels, most of which is spent in computation of weighted Delaunay triangulation and alpha complex. Note that these running times are for serial implementation of Delaunay triangulation and alpha complex computation. With GPU based implementation, we believe the running times will improve significantly.

3.3.2 Comparison with other channel extraction tools

We compare ChExVis web-server with four channel extraction tools *viz.* Mole 2.0 [87], Caver Analyst 1.0 [57], MolAxis [100] and PoreWalker [79]. Table 3.1 summarizes the features provided in these tools. As evident from the comparison table, the primary novelty of ChExVis is the advantage of extensive visual analysis of channels using channel profiles along with information-

[1] http://vgl.csa.iisc.ac.in/chexvis/

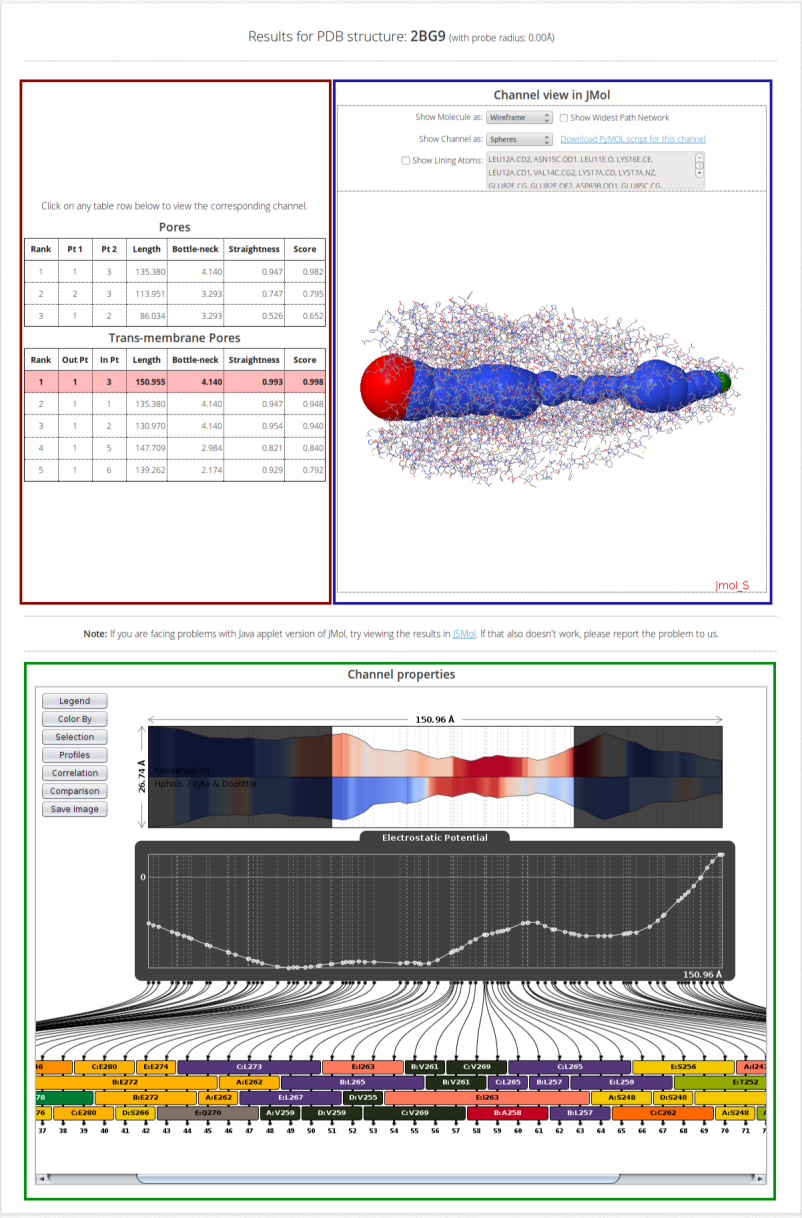


Figure 3.6: Screen-shot of output generated by ChExVis for PDB-id 2BG9. The output page has 3 main regions (coloured boxes shown only in this illustration). On top left (red box), the automatically computed channels extracted by ChExVis are listed along with their properties in tabular form. The user can select one of the rows of these tables to view that particular channel. On top right (blue box), the 3D view of the currently selected channel is shown in the context of the protein within JMol browser plug-in. At the bottom (green box), the 2D representation of the channel is shown in a Java applet called ChExVis properties viewer. Both the 3D and 2D views of the channel are highly interactive and provide rich visualization features. The views are linked to tables listing the channels. Selecting a different channel immediately updates the 2D and 3D views.

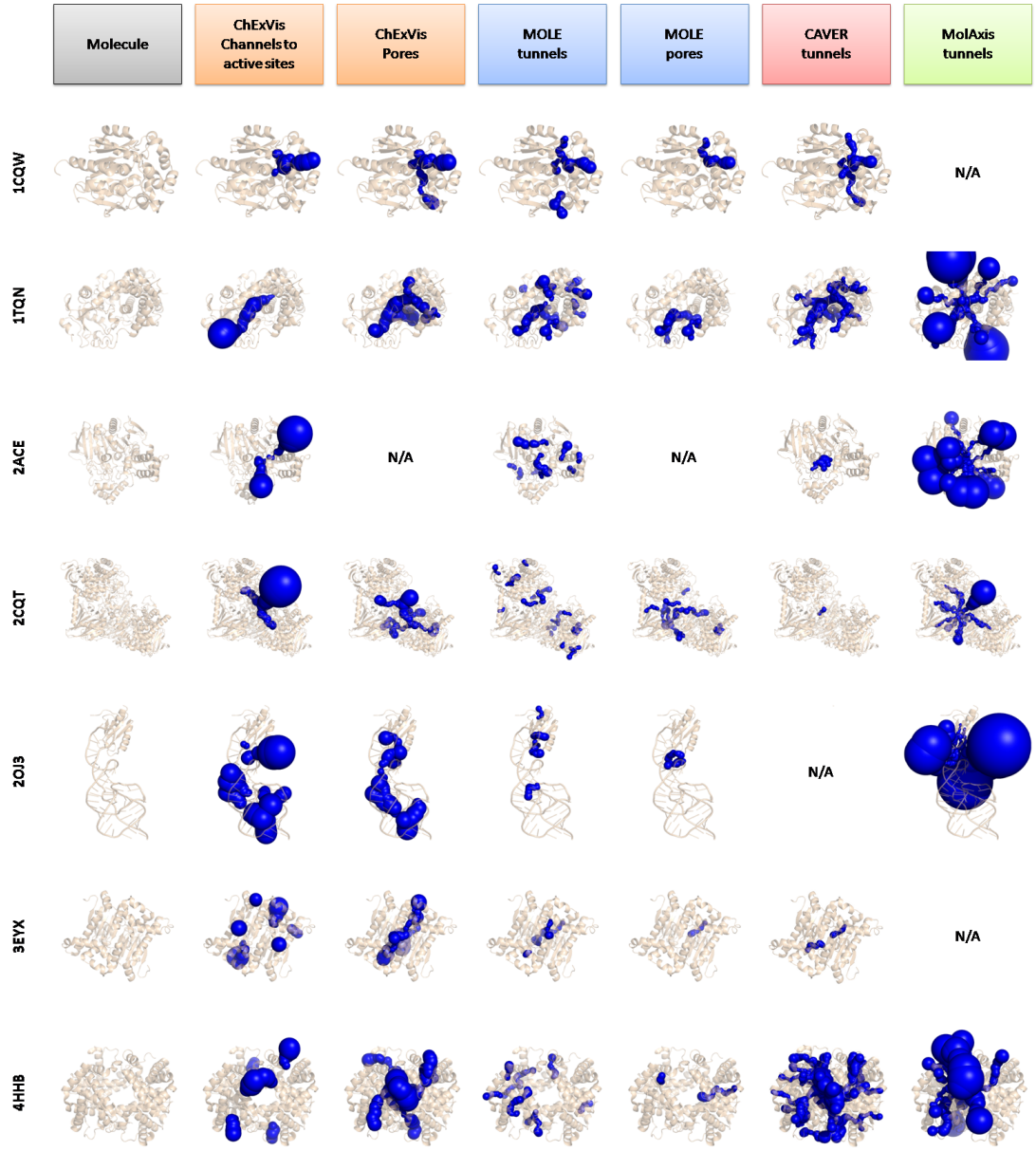


Figure 3.7: The results of extraction of channels in a select set of enzymes using different tools *viz.* ChExVis, Mole, Caver and MolAxis.

rich 2D representation of channels. However, we also claim that the proposed channel extraction method is significantly better than the other tools. To support our claim, we compared the results of these methods for a set of 29 diverse protein structures. The channels detected in the seven enzymes which are a part of our dataset, are summarized in Figure 3.7, while pores extracted in 22

Table 3.1: Comparison of features supported in different channel extraction tools

Feature	ChExVis	Mole	Caver	MolAxis	PoreWalker
Computation of channels leading to internal molecular sites	Yes	Yes	Yes	Yes	No
Automatic suggestion of internal sites using CSA	Yes	Yes	Yes	No	No
Channels to multiple internal sites	Yes	Yes	Yes	No	No
Filtering of similar channels	No	Yes	Yes	No	No
Computation of pores	Yes	Yes	No	Limited	Only transmembrane pores
Computation of transmembrane pores	Yes	No	No	Limited	Yes
Multiple transmembrane pores	Yes	No	No	No	No
Ranking of pores	Yes	No	No	No	Not applicable
Computation of physico-chemical properties	Yes	Yes	No	No	No
Interactive visualization of physico-chemical properties	Yes	No	No	No	No
Conservation profile	Yes	No	No	No	No
User Interface	Web-server	Web-server and GUI	GUI	Web-server	Web-server
Interactive visualization of channels	Yes	Yes	Yes	No	No
Good default parameters	Yes	Yes	Yes	Limited	Yes
Computation speed	Reasonable	Fast	Reasonable	Reasonable	Slow
PyMOL export	Yes	Yes	Yes	No	No
PyMOL plugin	No	Yes	Yes	No	No

transmembrane structures by different methods are shown in Figures 3.8, 3.9 and 3.10.

For comparison, we used the default parameters for channel extraction provided by the respective tool. ChExVis computes three types of outputs *viz.* channels leading to active sites, pores and transmembrane pores. Mole computes channels leading to active sites[1] as well as pores, while Caver computes only the former. In addition to computing channels leading to a site in the molecule, MolAxis can also extract transmembrane pore if the axis and a constraint sphere are provided. PoreWalker computes a single transmembrane pore. We compare ChExVis channels with the channels computed by Mole, Caver and MolAxis. ChExVis pores are compared with Mole pores. The top transmembrane pore computed by ChExVis is compared with the transmembrane pore computed

[1]Mole and Caver use the term *tunnel* to refer to a channel that leads to a site inside the molecule, *i.e.*, tunnels have one end point incident on a group of atoms buried inside the molecule while the other end point is close to the molecular exterior.

by MolAxis and PoreWalker.

3.3.3 Summary of comparison

The qualitative comparison of results produced by different tools using default input parameters are summarized in Figures 3.7, 3.8, 3.9 and 3.10. For the 22 transmembrane structures in our dataset, we also attempt a quantitative comparison of pores identified by these tools, by reporting the number of structures in which biologically significant pores are correctly identified. The biologically significant pores (the ground truth) are determined based on prior studies reported elsewhere in literature. See Table 3.2.

Table 3.2: Quantitative comparison of ChExVis with other tools in their ability to correctly identify transmembrane pores in 22 protein structures in our dataset.

Software	Correct	Partially Correct	Wrong
Mole	5	0	17
MolAxis	12	0	10
PoreWalker	17	3	2
ChExVis	19	3	0

Mole. Mole provides excellent stand-alone and web-based interactive interface for extraction of multiple channels. Mole utilizes the CSA database [45] and HETATM records to automatically identify catalytic sites in the protein. This facility is also supported by the ChExVis web-server. As evident from results in Figure 3.7, compared to ChExVis, Mole reports more tunnels leading to active sites. This is due to the fact that ChExVis reports a single best channel from each active site, while Mole extracts multiple tunnels and clusters the extracted channels to reduce the number of reported channels. The option to extract multiple channels from a single point of origin is available in a stand-alone version of ChExVis.

However, Mole does not support extraction of transmembrane pores. We used its pore extraction option to determine pores and compared the results against pores extracted by ChExVis. Among the 22 transmembrane proteins, using default parameters Mole fails to identify correct transmembrane pore in 17 structures. In comparison, ChExVis identifies correct transmembrane pores in 19 structures and partially correct pores in the remaining 3 structures (see Figures 3.8, 3.9 and 3.10). Figure 3.11 compares the results obtained by Mole and ChExVis in the PDB structure 3EAM. Mole fails to identify the long and wide pore going across the membrane, and instead identifies a few side channels. ChExVis, on the other hand, correctly identifies the biologically significant pore [8] as the top ranked transmembrane pore. It also identifies some of the side channels

identified by Mole. Figure 3.12 demonstrates another example where Mole does not report correct transmembrane pores in 2J1N.

Caver. Caver is a command-line based channel extraction tool. A commercial GUI front-end of this tool was released recently as Caver Analyst 1.0 [57]. We used the evaluation version of Caver Analyst 1.0 for the experiments. Caver Analyst supports automatic identification of active sites using CSA database. However, the number of origin points in a particular run is limited to one. The tunnels extracted by Caver are reported in Figure 3.7. Qualitatively, the results reported by Caver are not better than those reported by ChExVis. Caver was not used for comparison of pores extracted in transmembrane proteins, as it does not provide special support for extraction of pores in general and specifically transmembrane pores.

MolAxis. MolAxis uses a Voronoi-based approach for channel detection. It automatically uses the center of largest cavity as the origin point of the channel. This is a useful heuristic but this single point may not correspond to an active site. Further, channels leading to multiple sites cannot be extracted in a single run. As shown in Figure 3.7, MolAxis fails to identify channels leading to active sites in two of the seven enzymes. MolAxis also provides an option of extracting a transmembrane pore, but it requires specification of the pore axis and a guiding sphere, which requires prior analysis of the protein and smart inputs by the user. As a result, it is difficult to identify transmembrane pores using MolAxis. When reasonable default values are chosen for pore axis and a guiding sphere, we observed that MolAxis computes correct transmembrane pores in 12 out of 22 transmembrane proteins (see Figures 3.8, 3.9 and 3.10). In cases of success, the extracted transmembrane pore mostly agrees with the top transmembrane pore detected by ChExVis.

PoreWalker. PoreWalker is the state-of-the-art tool specially designed for extraction of transmembrane pores. Given a transmembrane protein, PoreWalker reports the best transmembrane pore and residues lining that pore. This tool correctly identified biologically significant pore in 17 out of 22 structures while partially correct pores in 3 other structures (see Figures 3.8, 3.9 and 3.10). However, it was unsuccessful in identifying correct pores in two structures in our dataset. One such example is reported in detail in Figure 3.12. Here, instead of identifying any of the three biologically significant pores through the beta-barrels, PoreWalker incorrectly identifies the empty space between three units as the most significant transmembrane pore. Moreover, PoreWalker is designed to report only one transmembrane pore. So, in cases such as 2J1N (Figure 3.12), PoreWalker does not report all three transmembrane pores, which are equally important. ChExVis overcomes this problem by reporting multiple transmembrane pores, and in this example correctly identifies all three pores as shown in Figure 3.12(e).

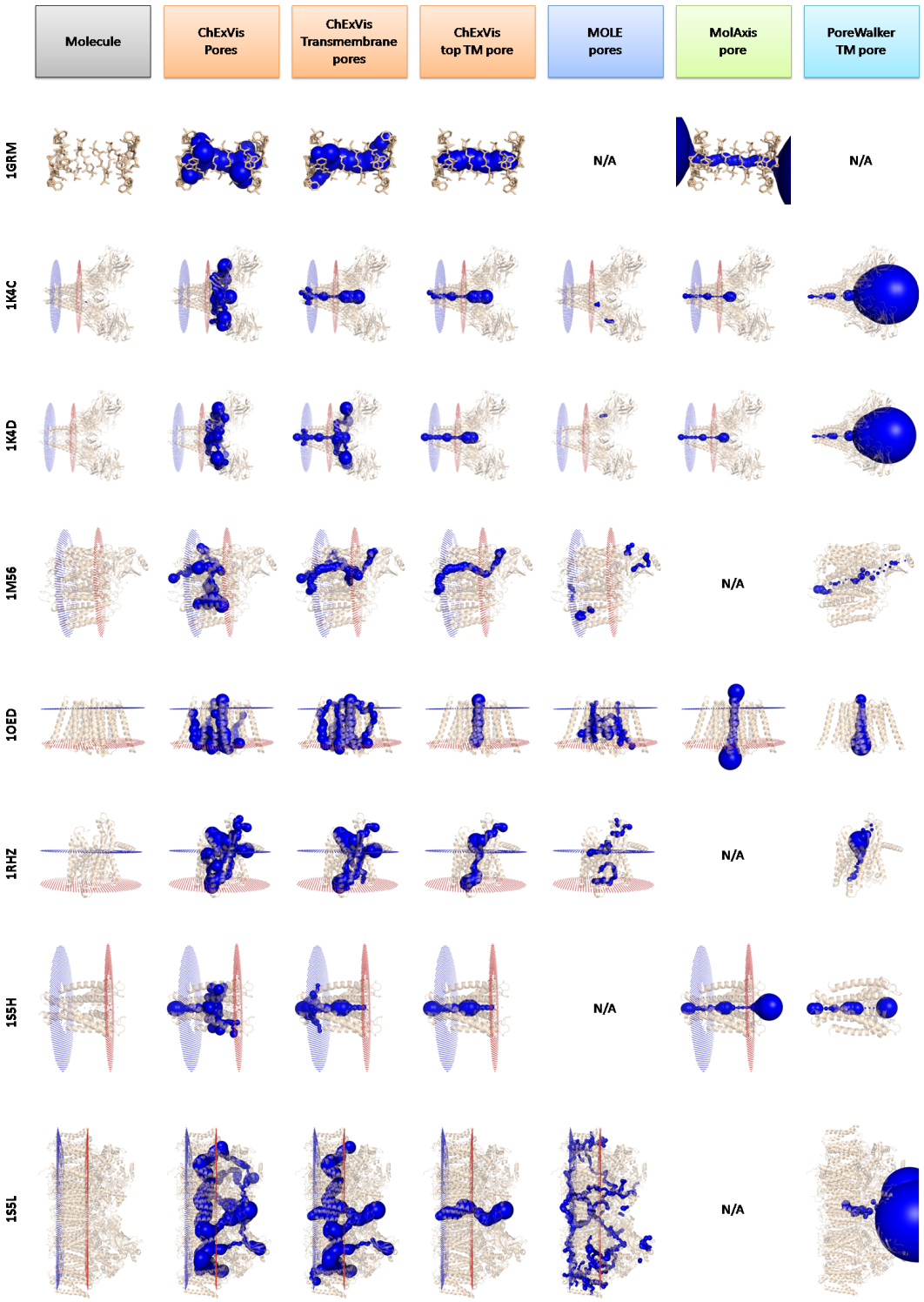


Figure 3.8: The results of extraction of pores in transmembrane proteins using different tools *viz.* ChExVis, Mole, MolAxis and PoreWalker.

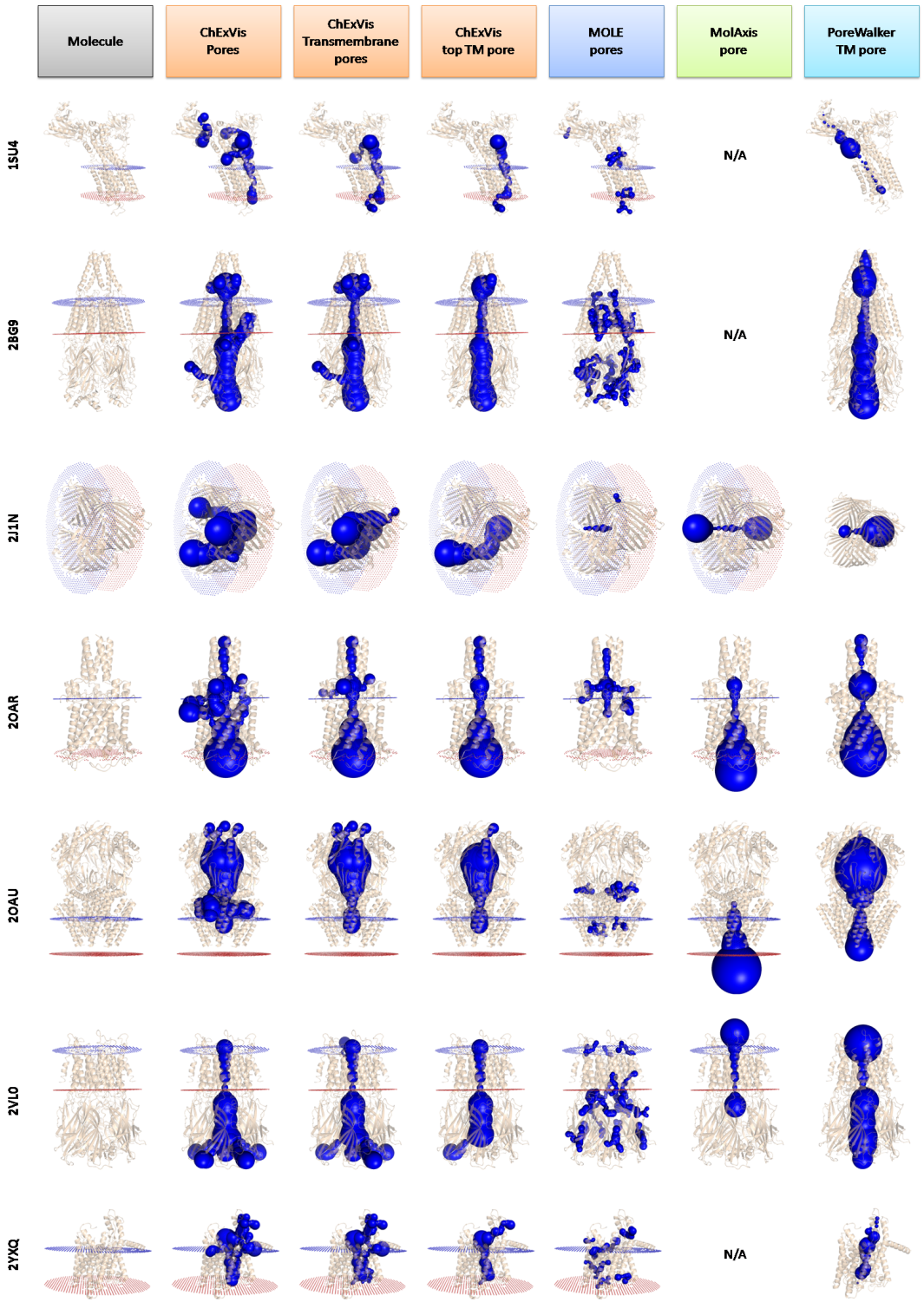


Figure 3.9: The results of extraction of pores in transmembrane proteins using different tools *viz.* ChExVis, Mole, MolAxis and PoreWalker.

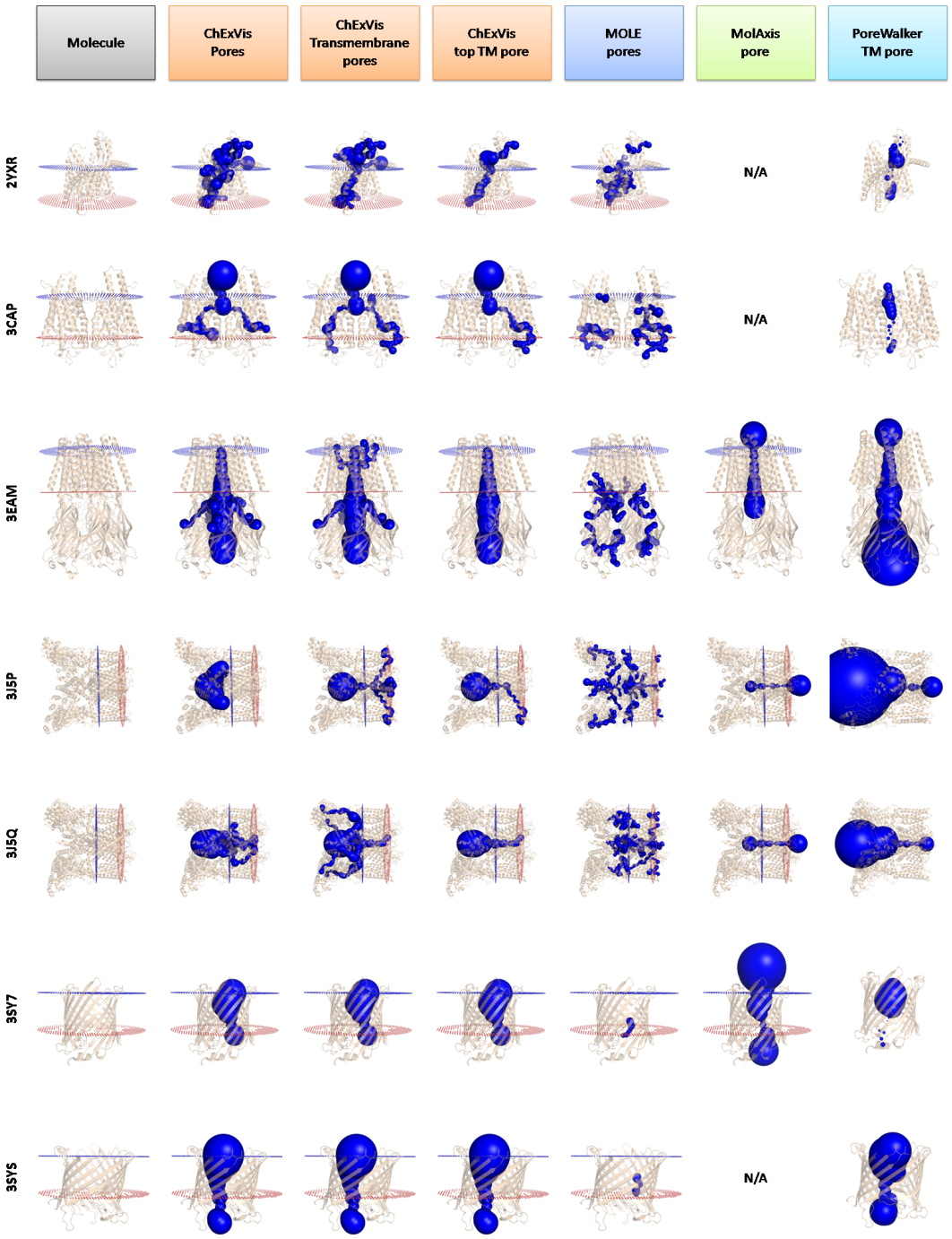
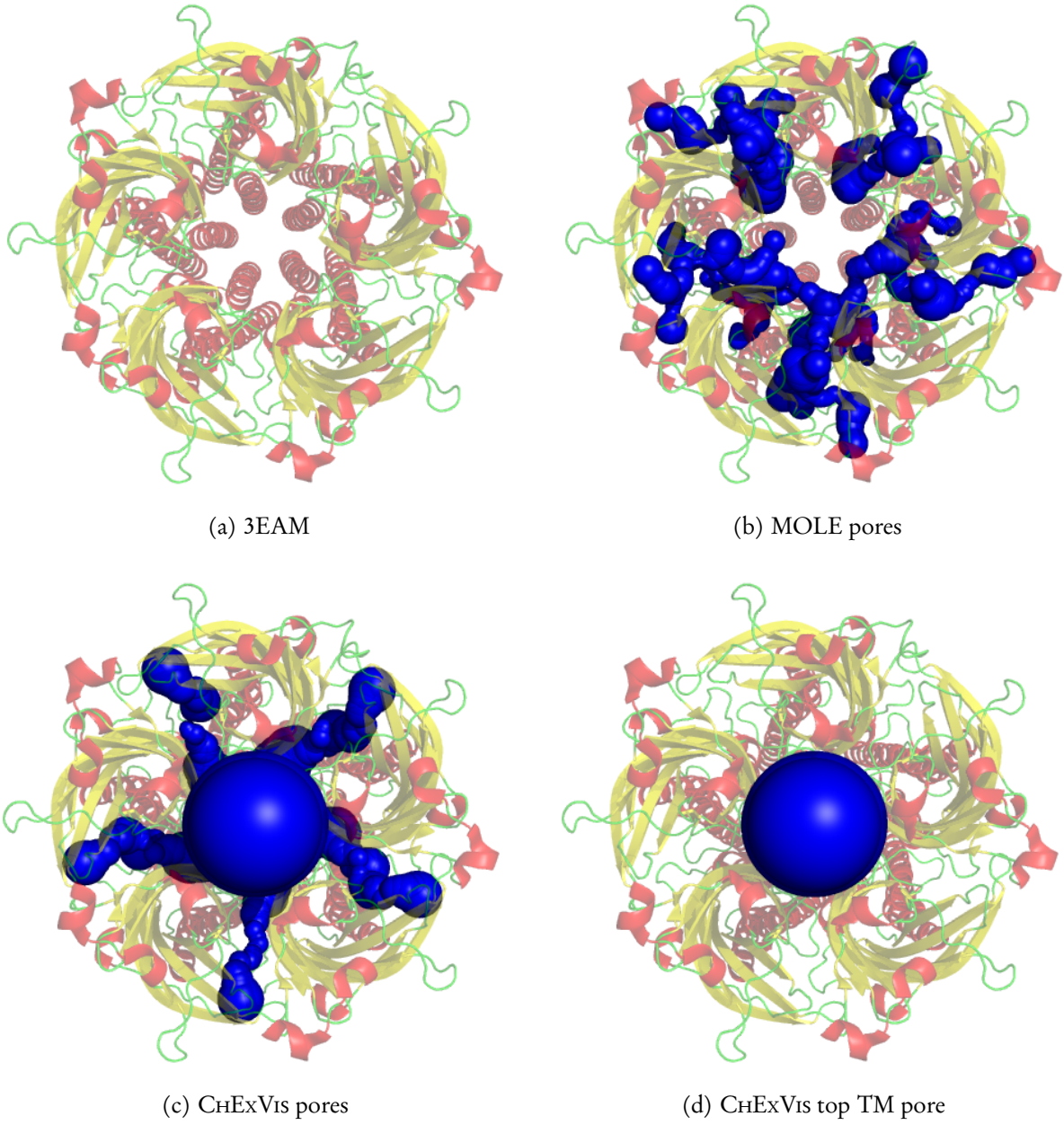


Figure 3.10: The results of extraction of pores in transmembrane proteins using different tools *viz.* ChExVis, Mole, MolAxis and PoreWalker.



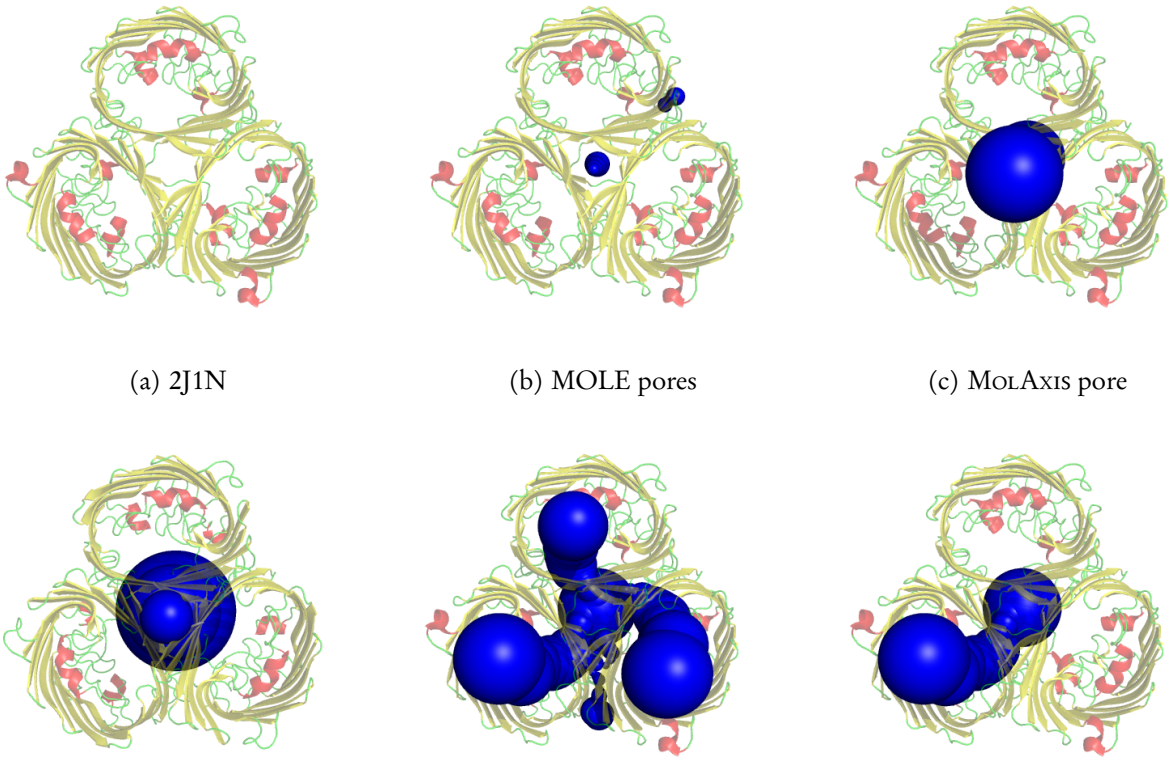
(a) 3EAM

(b) MOLE pores

(c) ChExVis pores

(d) ChExVis top TM pore

Figure 3.11: Comparison of pores extracted by Mole and ChExVis in 3EAM. (a) Cartoon representation of the transmembrane protein 3EAM. The 3D view is such that the transmembrane pore is perpendicular to the plane of the page. (b) The pores extracted by Mole in this structure. Mole identifies a few side channels going from central pore to outside, but it fails to identify the transmembrane pore. (c) On the other hand, ChExVis identifies the main transmembrane pore as well as some side channels. (d) Also, the top transmembrane pore suggested by ChExVis is verified to be the correct transmembrane pore in this structure [8].



(a) 2J1N (b) MOLE pores (c) MolAxis pore

(d) PoreWalker result (e) ChExVis pores (f) ChExVis top TM pore

Figure 3.12: Comparison of pores extracted by Mole, MolAxis, PoreWalker and ChExVis in 2J1N. (a) Cartoon representation of the transmembrane protein 2J1N. This structure consists of three beta-barrel subunits going across the cell membrane. The figure shows top view of the protein such that cell membrane is parallel to the plane of the page. (b) Mole identifies two pores in this structure, both of them are not correct transmembrane pore passing through beta-barrels. Mole wrongly identifies the narrow space between the three units as a transmembrane pore. (c) MolAxis also identifies a pore going through space between three subunits. This maybe due to the specified input parameters. (d) PoreWalker also fails to identify the correct transmembrane pores in this structure. It identifies the empty space between three subunits as the only transmembrane pore. By design, PoreWalker would not have been able to identify all the three transmembrane pores as it extracts only one transmembrane pore. (e) ChExVis is able to correctly identify transmembrane pores through all the three subunits, using default parameters for finding pores. (f) Unlike other tools which identify pore through the space between subunits, the top transmembrane pore identified by ChExVis is one of the pore passing through a beta-barrel subunit.

3.4 Applications to the analysis of channels in proteins

The 2D and 3D visualization of the channel, coupled to a display of its various properties of the channel through profile plots, is an important feature and advancement over existing channel visualization tools. The biochemical layout of the channel lining residues and capture of their physico-chemical properties such as radius, volume and solvent accessibility are useful in appreciating the channel type and its lining residues. A novel feature is the inclusion of conservation scores derived through comparison of residue propensities in related channels using the ConSurf web-server [3]. This is useful in evaluations of the observed mutability amongst closely related homologues and in evaluating residues that may be substituted in experimental mutagenesis, or in guiding the decisions on the substitutions that are likely to be tolerated. Further, the profile layouts help evaluate the importance and contribution of the observed amino acids at various positions, to appreciate interactions that are observed in the channel and are crucial to its function. With the following examples we describe some applications of ChExVis in the detection of channels and its use in appreciating the function of the channel at various levels of detail.

3.4.1 Comparison of open and closed states of the protein

Pentameric ligand-gated ion channels. Pentameric ligand-gated ion channels constitute a large family of ionotropic channels that are ubiquitously represented and fairly conserved in the animal kingdom [86]. The 3D structures of a number of such channels were submitted to ChExVis to derive pore features. A homo-pentameric organization that is constituted by a highly conserved extracellular domain folded as a beta-sandwich and a helical transmembrane domain [24], is captured in all the queried proteins. We compared profile views of the ligand-gated ion channel in its closed (2VL0:ELIC) and open states (3EAM:GLIC), to determine if ChExVis could capture the similarities and differences in these states through analysis of features such as: a) the changes in the diameter of the channel, b) accessible solvent area / volume of the channel, c) residues lining the channel, and, d) physico-chemical properties of residues lining the channel. As seen in the profile views, in both states, pore-interfacing residues from all the five chains lie in fairly equivalent positions. The 2D profile plots of the residue contributions to the channel show that the side chain atoms of the channel-lining residues point into the channel lumen (see Figures 3.19 and 3.20). Differences in the open (3EAM:GLIC) and non-conducting closed conformation of the channel (2VL0:ELIC) include a constriction lined by hydrophobic residues, possibly contributing to ion-selectivity and corresponding to a bottle-neck radius of 2.21Å while this is 5.12Å in the open state (3EAM:GLIC) [26]. Labelled view of the profiles show that F246 in 2VL0 lies at the narrowest region of the channel, as reported elsewhere [19]. The open and closed states of the channel also

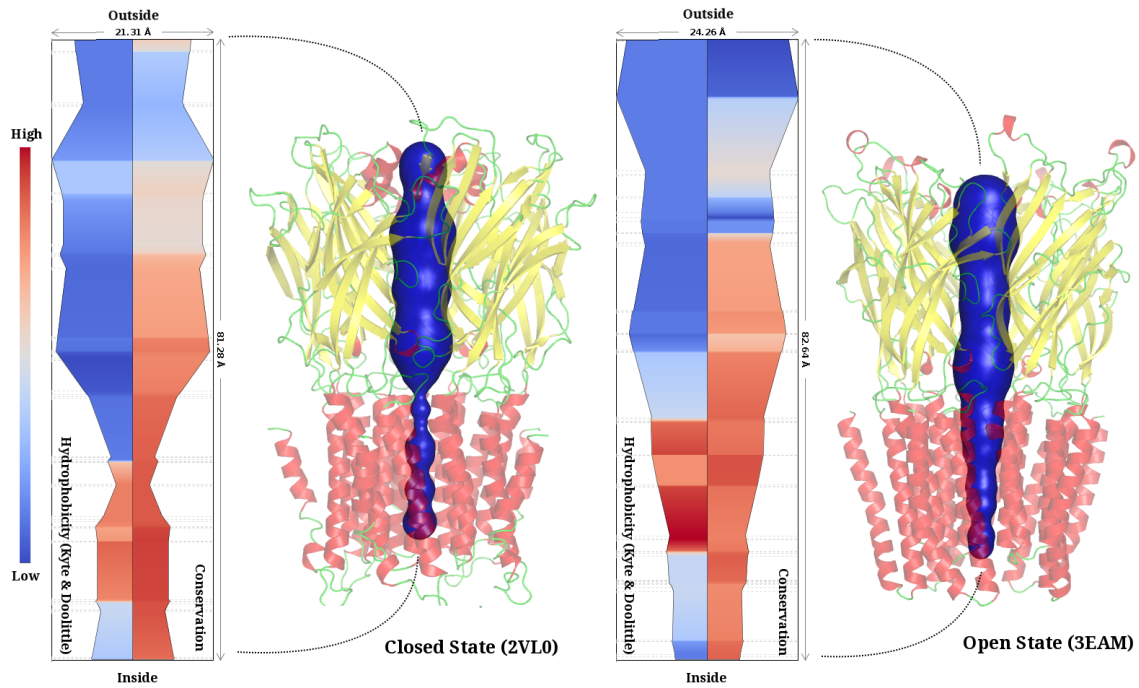


Figure 3.13: The pentameric ligand-gated ion channels extracted in closed and open conformations. The channels are wide and hydrophilic in extra-cellular region which is less conserved. The constricted transmembrane spanning helical channel section is hydrophobic and highly conserved. The bottleneck radius is 2.21Å in closed configuration while it is 5.12Å in open state.

show marked differences in their relative solvent accessibility, as reported through their DSSP accessibility scores. The pore opening in 3EAM is lined predominantly by hydrophilic residues and subsequently followed by hydrophobic residues (Figure 3.13), in line with the structural findings of a 12Å wide extracellular hydrophobic vestibule and a funnel shaped transmembrane-spanning helical region. The conducting conformation in 3EAM, where the hydrophobic constriction has opened to an aqueous funnel-shaped channel is also captured [50].

ChExVis also correctly identifies the tetrameric assembly of the Transient Receptor Potential (TRP) channel and lists residues lying on the outer pore, central canal, and selectivity filter. As seen in Figure 3.14, the entry to the channel is a wide funnel-shaped pore in both the closed (3J5P) and open (3J5Q) states of the channel. The residue-label view shows that the narrowest region of the channel is lined by residues G683, I679 and Y671 with their side chains pointing into the channel. I679, the bulkiest residue in the lower gate of the channel, from each subunit comes together to form the most constricted point in the gate that seals off further access to the channel. In line with structural findings, its conservation scores shows that it is conserved in all homologues [67]. The radius of the hydrophobic seal in the lower gate changes from 2.6Å to 4.7Å in the two struc-

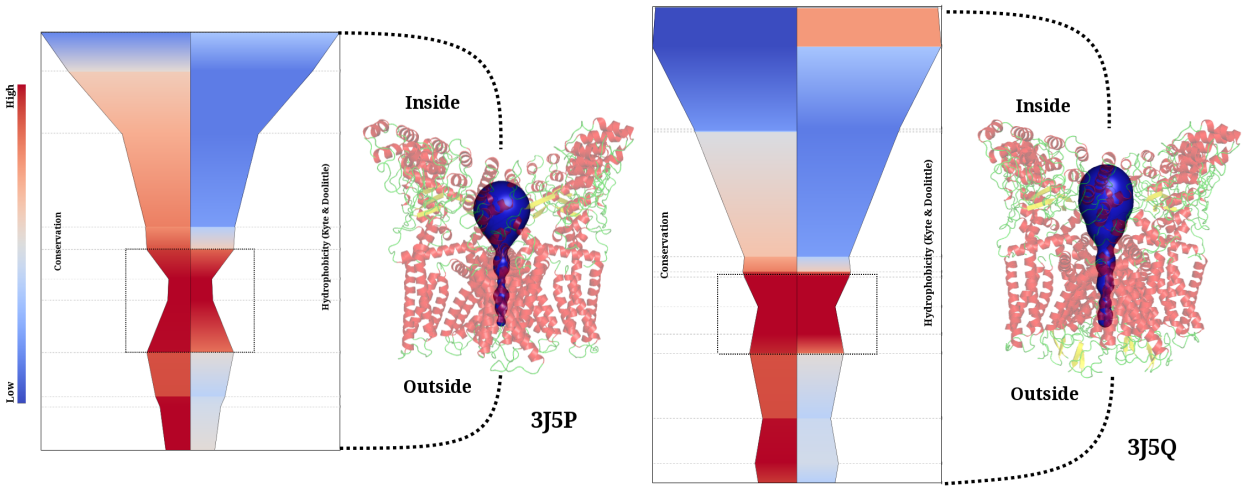


Figure 3.14: Transient Receptor Potential (TRP) channels extracted in closed and open state. The constricted region of the channel (highlighted by rectangles) is both highly conserved and hydrophobic. The bottleneck radius is 2.6Å in closed configuration while it is 4.66Å in open state.

tures. Clearly, although lining residues do not change remarkably, side chains re-orient as a result of gating, resulting in a wider pore in the open state of 3J5Q. See Figures 3.21 and 3.22 for detailed comparison of channels).

3.4.2 Comparison of channels exhibiting wide substrate specificities

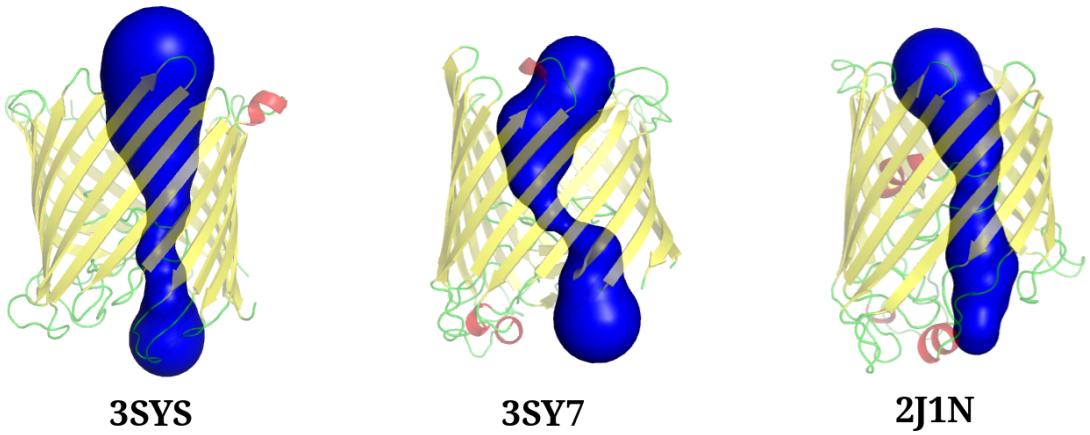


Figure 3.15: Outer Membrane (OM) carboxylate channels in structures 3SYS and 3SY7. These channels are narrower compared to channel porin (2J1N) indicating that they are more selective.

The outer membrane proteins of several gram negative bacteria enable uptake of nutrients either through non-specific channels/porins and therefore, at high substrate concentrations, or in a

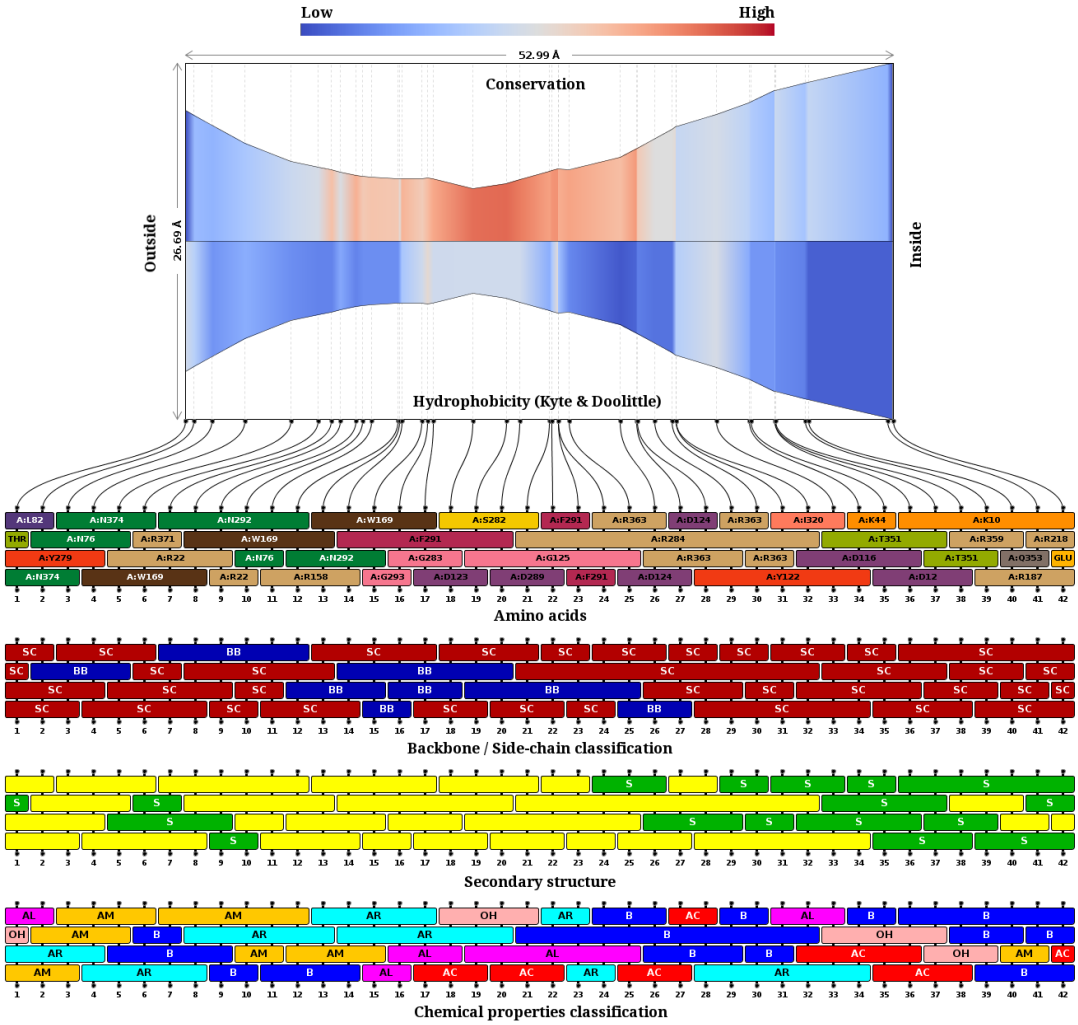


Figure 3.16: Properties of membrane carboxylate channel in 3SYS. The 2D profile is coloured by conservation and hydrophobicity of residues. It can be observed that the channel constriction is more conserved than the rest of the channel. According to box representation of first row, the amino acids lining the channel shows higher proportion of Arginine. Most of the residue side chains point towards the channel as can be concluded by many red boxes in the second row. The third row shows that channel is surrounded by loops (coloured yellow), specially at the constriction. Although some residues at channel end points belong to outer beta-barrel structure. Lastly, the fourth row shows the chemical properties of the residues lining the channel. The basic residues which play an important role in the function of this channel are correctly identified by ChExVis. They are represented as dark blue boxes.

substrate-specific manner through outer membrane carboxylic acid channels (Occ family). Since the Occ channel proteins are dedicated to the uptake of wide variety of substrates, we derived the pore features for different members of this family (3SY7, 3SY9, 3SYB, 3SYS, 3SZD, 3SZV, 3T0S,

3T20 and 3T24) and compared them to determine if the visual capture of the channel properties through 2D and 3D views could distinguish their various sub-types. Comparisons of the two OM channels, OccD1 (3SY7) and OccK1 (3SYS) shows that both channels form monomeric barrels around a central channel that is constricted at a region lined predominantly by basic residues such as arginine and lysine (see Figures 3.15, 3.16 and 3.23). Four of the seven residues forming the basic ladder in this class of proteins (OccK1:3SYS) are captured by ChExVis while projections of the channel neighbourhood capture other basic residues that are typical of the family (R22, R126, R158, R324, R306, R381 and R363). The plots also show that the constricted regions are contributed by loops and lined by residues that are poorly conserved in related homologues except for regions in the immediate vicinity of the constriction. Pore size can dictate the size of the substrate that can pass through the channel [70]. The differences in the pore size of the two amilies, with larger pore size observed in OccK family members to recognize a wider range of mono-cyclic substrates such as glucoronate and aromatic amino acids is correctly captured [44]. Contrastingly, the OccD family proteins have smaller pores that are highly specific and prefer smaller amino acids such as arginine. We also compared the Occ channels with porins (2J1N), which are non-specific channels with larger pores, and observe that the pore size of Occ family protein is indeed small and selective for small molecules [44]. We anticipate that visualization tools such as ChExVis that capture various views and properties of the residues lining the channel will add to knowledge on channel specificity and residue layout within the pore and a better appreciation of the differences between seemingly related molecules.

3.4.3 Comparison of channels in homologues and the impact of residue mutations on channel properties

Voltage-activated cation channels are membrane proteins that selectively conduct K^+, Na^+ and Ca^{2+} ions in response to changes in membrane voltage. The KcsA potassium channel contains two transmembrane segments and a P-loop signature that harbours a selectivity filter, which selects for the conductance of K^+ over Na^+ ions. Residues with a role as the filter from the four subunits, project into the channel centre to form a narrow pore. The K^+ ion-binding sites formed by residues G79, Y78, G77, V76 and T75, that form the ion binding site and square anti-prism around K^+ [91], are consistently identified by ChExVis in all the fourteen KcsA channels in the dataset (see Figure 3.17). Although the path into the cavity is not identified by the method, the rest of the channel and a large central cavity ($\approx$ 10Å), on the intracellular side, is correctly defined. Further, comparison of the structures at low (1K4D) and high salt (1K4C) concentrations shows that the conformation of the channel changes and moves from a closed to open state with increasing salt concentration (Figure 3.18). G77 contributes to channel constriction in 1K4D with a channel radius of 2.01Å. In

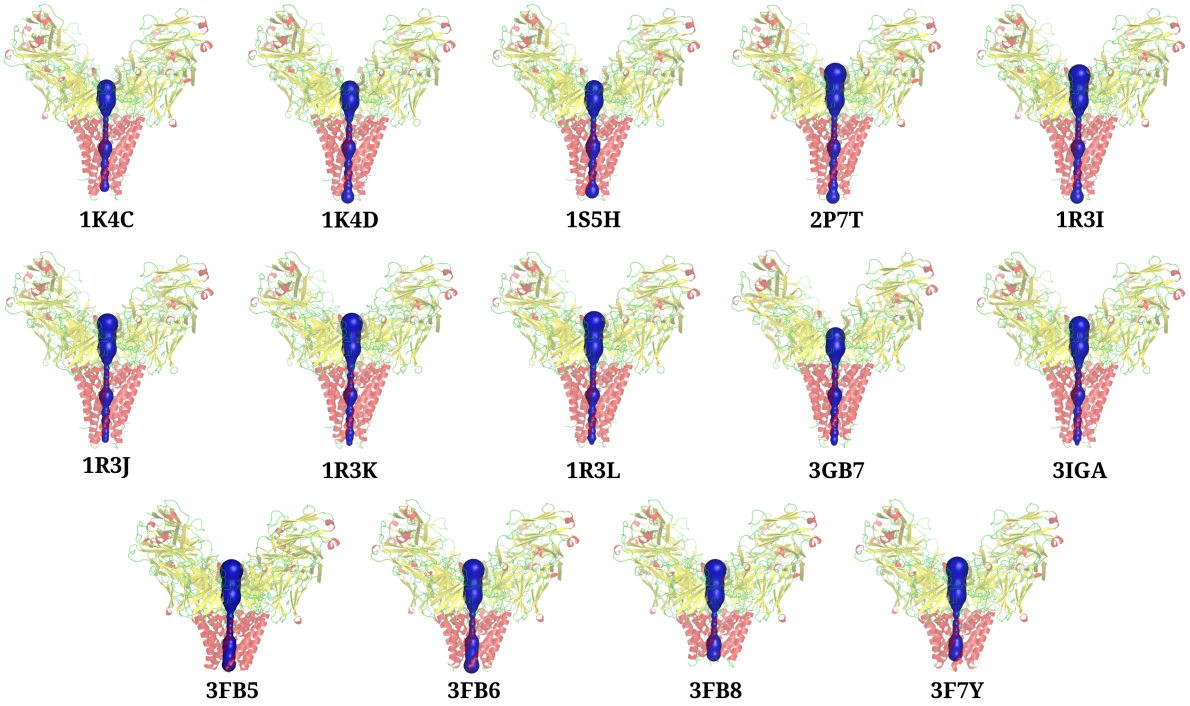


Figure 3.17: KcsA channels extracted in different structures by ChExVis. The PDB ID of the structures are shown below each figure.

1K4C, the open state, V76 also lines the channel and coordinates the K^+ ion through its carbonyl oxygen. Replacement of T75 with C75 in the mutant channel [102] also results in a widening of the channel, changing the bottle-neck radius from 2.26Å to 2.55Å. The mutant channel has low K^+ conductivity because of the absence of crucial oxygen atoms. This minor but critical change is captured by ChExVis visualizations (Figures 3.18 and 3.24).

3.5 Conclusions

We developed a new method for the extraction, visualization, and visual exploration of channels in biomolecules through a software tool, ChExVis, that is available as a web server. The method is automated and accepts 3D co-ordinates of the structure or its PDB ID as input. Channel location in the protein may be guided using bound ligand information, CSA, or be specified by the user. Here, channels are represented as a set of connected tetrahedra derived from the alpha complex. This representation enables efficient computation and interactive visualization of the channels together with different geometric and physico-chemical profiles.

The availability of visualization tools such as ChExVis that can identify multiple channels will be useful to understand the properties of the multiple channels detected in the protein family of

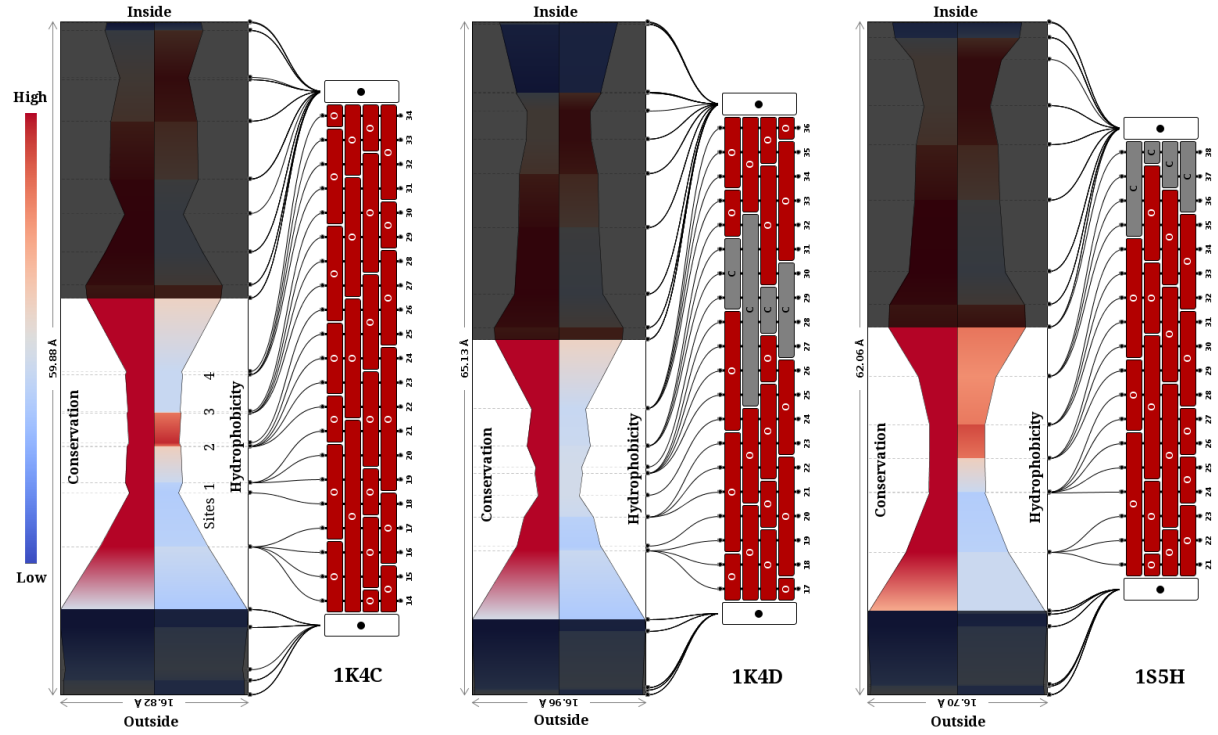


Figure 3.18: The KcsA potassium channels extracted in structures 1K4C, 1K4D and 1S5H. On left, channel extracted in 1K4C (high K^+ concentration) is shown. The channel has four highly conserved K^+ sites which are surrounded by carbonyl Oxygen atoms as shown in the profile shown. The channel closes in low K^+ concentration (1K4D) as captured by next profile. The channel in this structure is more constricted and one of the site is surrounded by Carbon atoms instead of carbonyl Oxygen atoms. Lastly, on right a mutant channel (1S5H) is shown which has reduced K^+ conduction capability. This is attributed to replacement of Oxygen atoms with Carbon atoms at crucial site no. 4, which is correctly captured by ChExVis in the profile shown.

interest and the multiple routes that might be available for regulation of protein function in the family. The method is shown to perform well using a number of examples. As with all channel visualisation tools, manual intervention and intuition are vital to assess the importance of the reported channels. Examples of its application in understanding the biological function of the protein at various levels such as in the comparison of open and closed states of the protein, or in appreciating the wide substrate-specificities of the channel or comparison of channels with homologues have already been shown here. Further, the method lists the top channels for the protein and also reports all channels observed in the protein. Such studies coupled with information on conservation within the protein family and mutability of residues lining the channel will further the understanding of the basic biology of transport through membrane proteins.

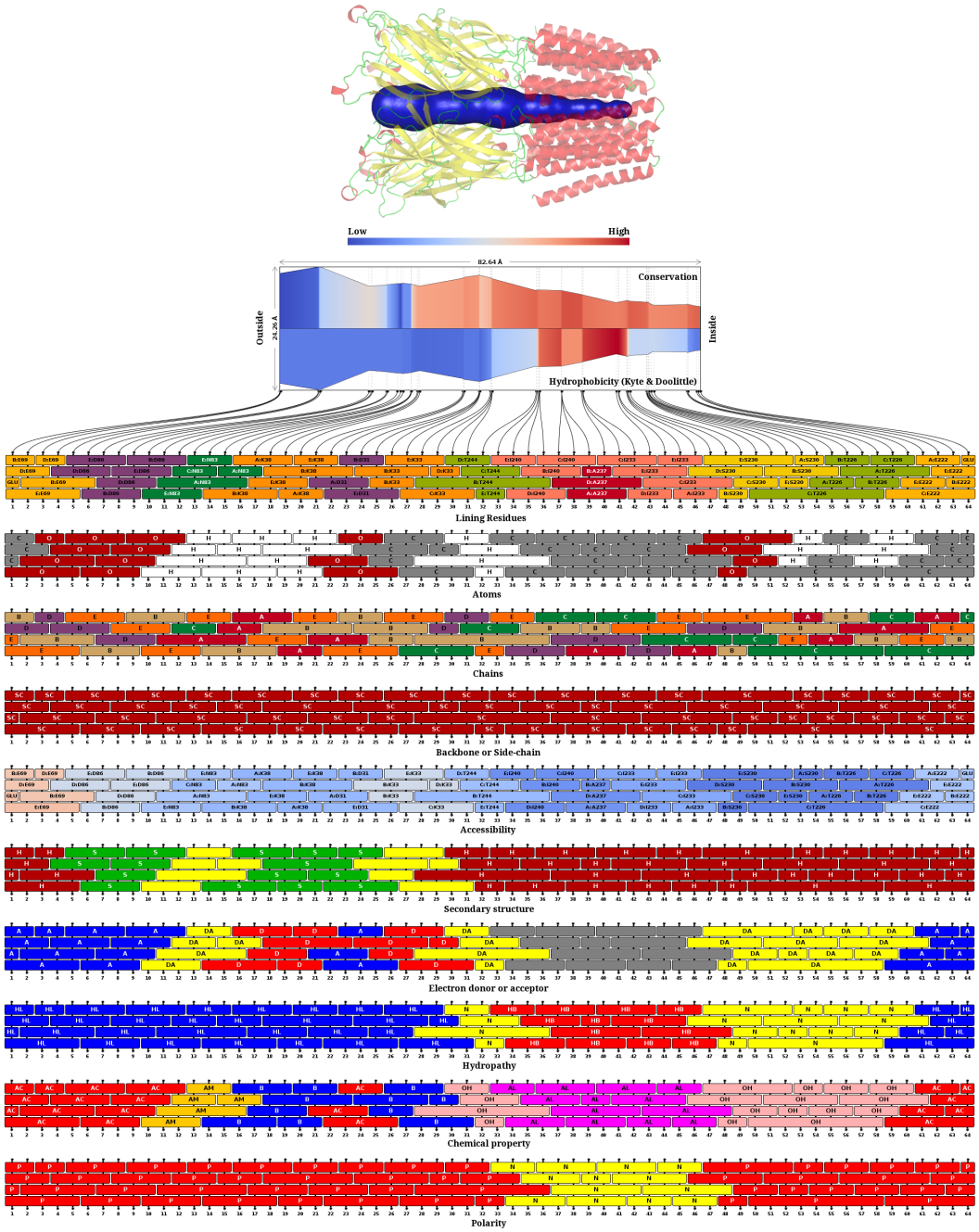


Figure 3.19: Properties of pentameric ligand-gated ion channel in the open structure 3EAM. The plots show residues from all five chains that interface with the pore in equivalent positions. The wall of the pore is lined primarily by the side chain atoms of various residues. The view shows that pore interior from positions 32-58 is least accessible while the positions 1-30 are solvent accessible. Channel positions 34-47 lined by residues I240, A237, I233, S230 are hydrophobic and lie on alpha-helices that correspond to the helical wall of the transmembrane spanning regions in the protein.

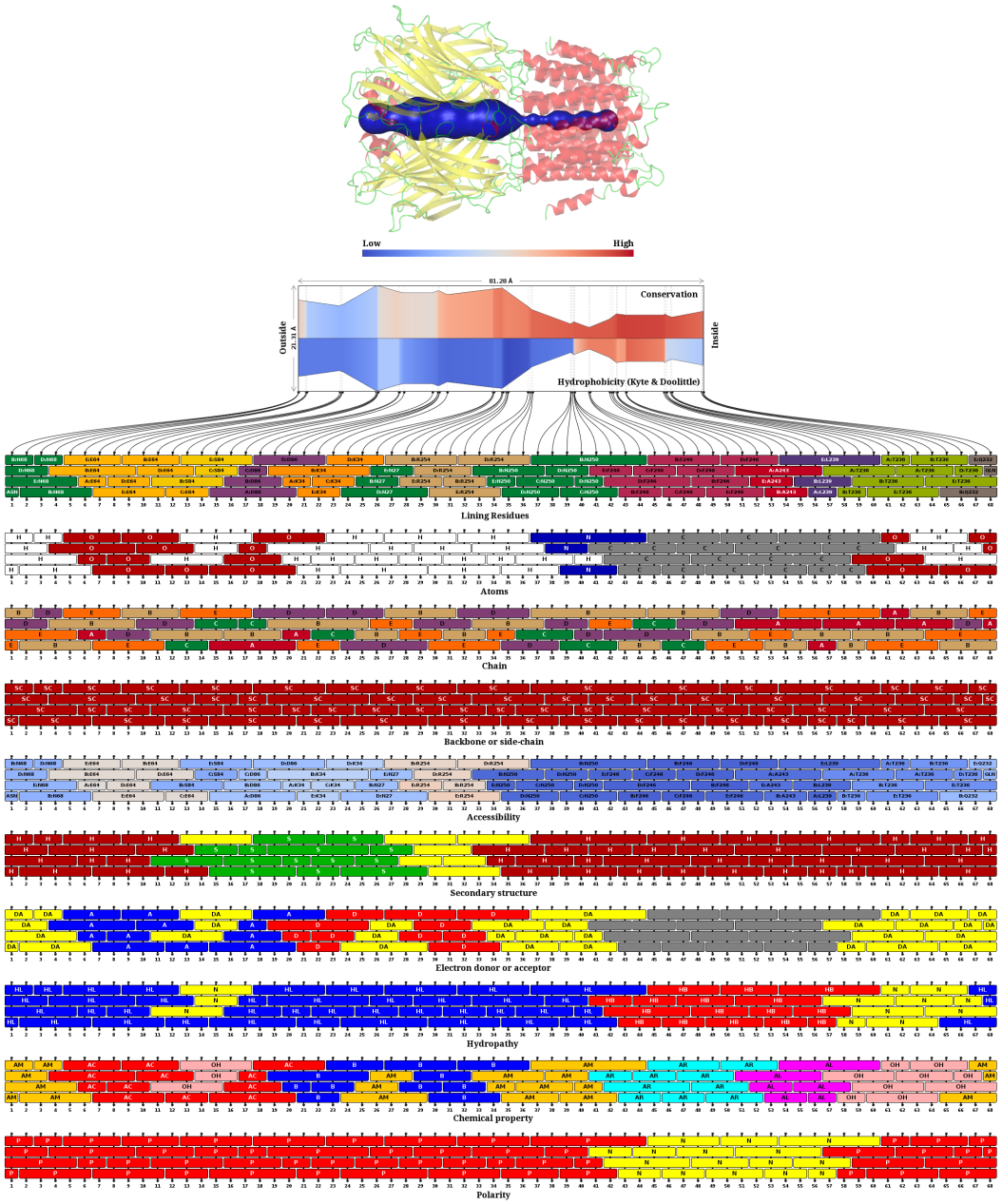


Figure 3.20: Properties of pentameric ligand-gated ion channel in the closed structure 2VL0. Compared to channel in open state (3EAM), the channel in 2VL0 is much more constricted at the transmembrane helical region. There is also a drastic reduction in solvent accessibility of this region from 40Å^2 to 10Å^2. Some changes in chemical properties of the the lining residues are also observable in closed conformation, as N250 and F246 protrude towards the pore around constriction as against I240 and I233 in open state. Also, in these channels the extra-cellular tunnel lined by hydrophilic residues is not very conserved, while the narrow selective region lying in transmembrane region is highly conserved.

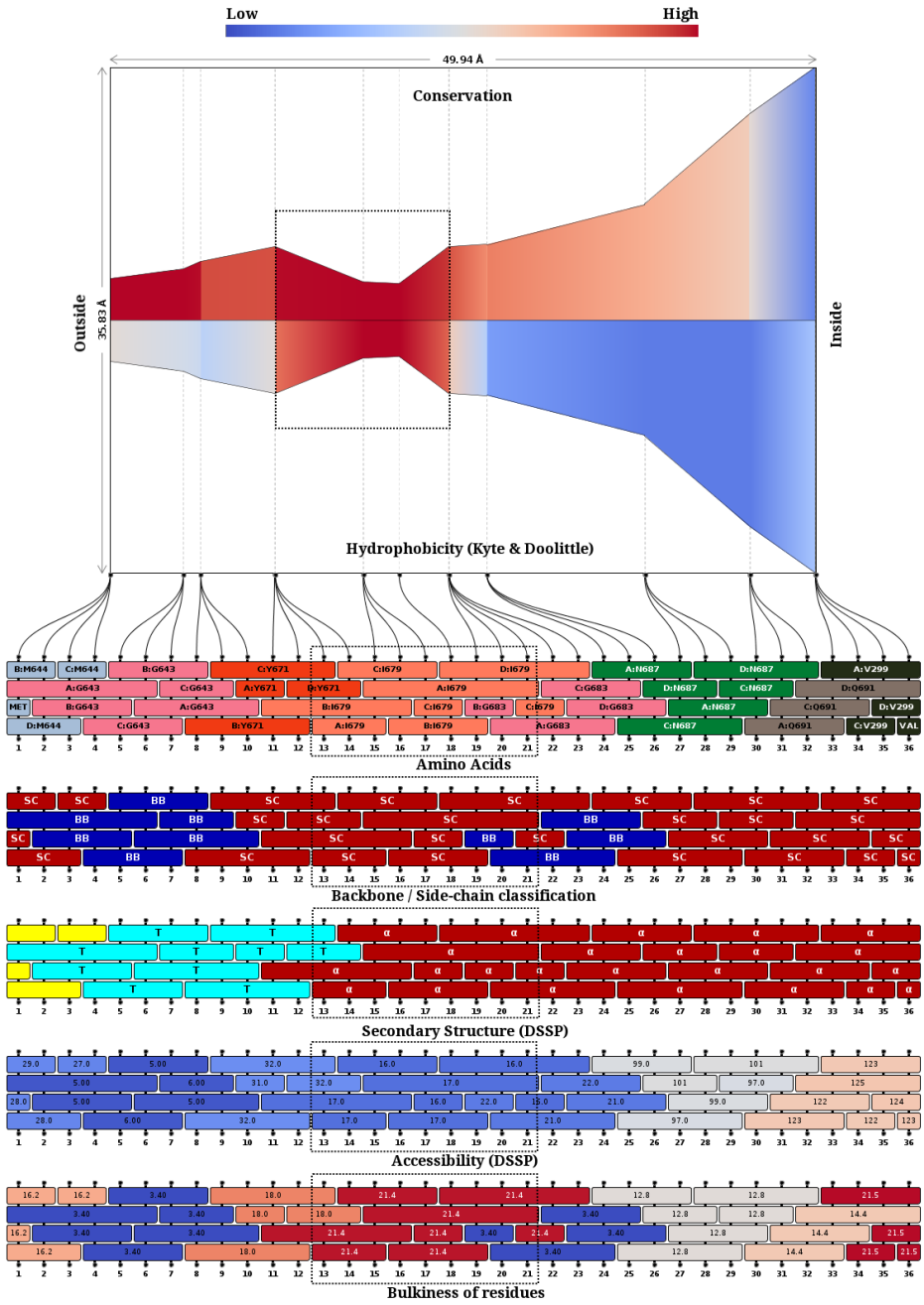


Figure 3.21: Properties of transient receptor potential channel in the closed structure 3J5P. In this structure, the radius of the channel is only 2.6Å at the narrowest point. The 2D profile is coloured by conservation and hydrophobicity, red color denoting high values while blue denotes low values. It can be observed the constricted region is highly conserved and hydrophobic. The next five rows of box representations show the properties of amino-acids lining the channel. From this data, it can be concluded that I679 lies at the narrowest point of the channel. Further I679 is a bulky residue with side-chain protruding towards the channel and it has low accessibility. According to third row, most of the residues lining the channel are part of alpha-helices and small helical turns towards the end.

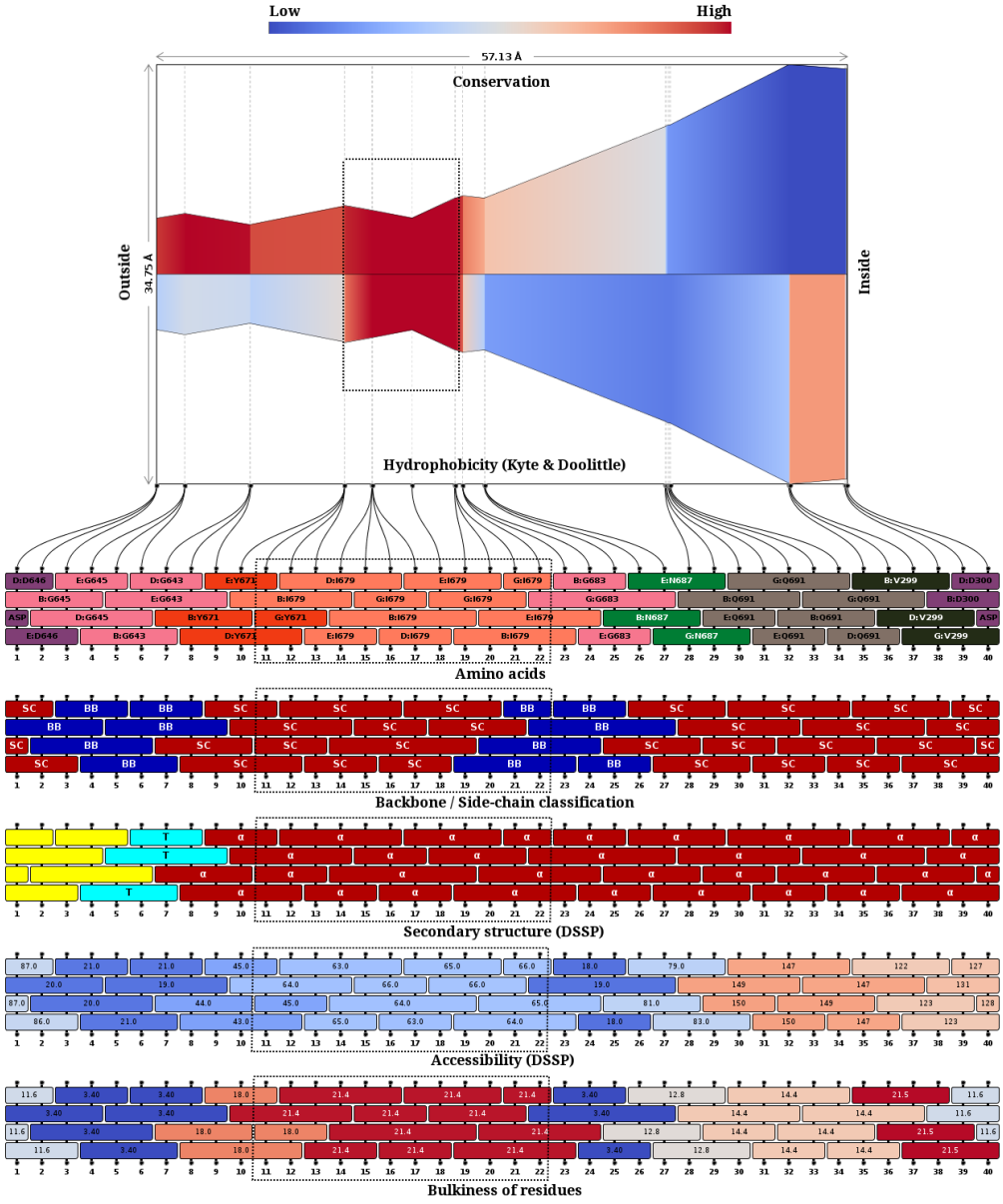


Figure 3.22: Properties of transient receptor potential channel in the open structure 3J5Q. In this structure, the radius of the channel is 4.66Å at the narrowest point, which is substantial improvement over closed structure bottleneck radius of 2.6Å. The 2D profile is coloured by conservation and hydrophobicity, revealing that the constricted region is highly conserved and hydrophobic. It can be seen most of residues lining the channel are same as that in 3J5P. Again I679 is an important residue lying at the constriction point. It is important to observe that accessibility of the channel residues in 3J5Q is higher than in closed state i.e. 3J5P. Specially, accessibility of I679 goes up from 16Å^2 to 65Å^2.

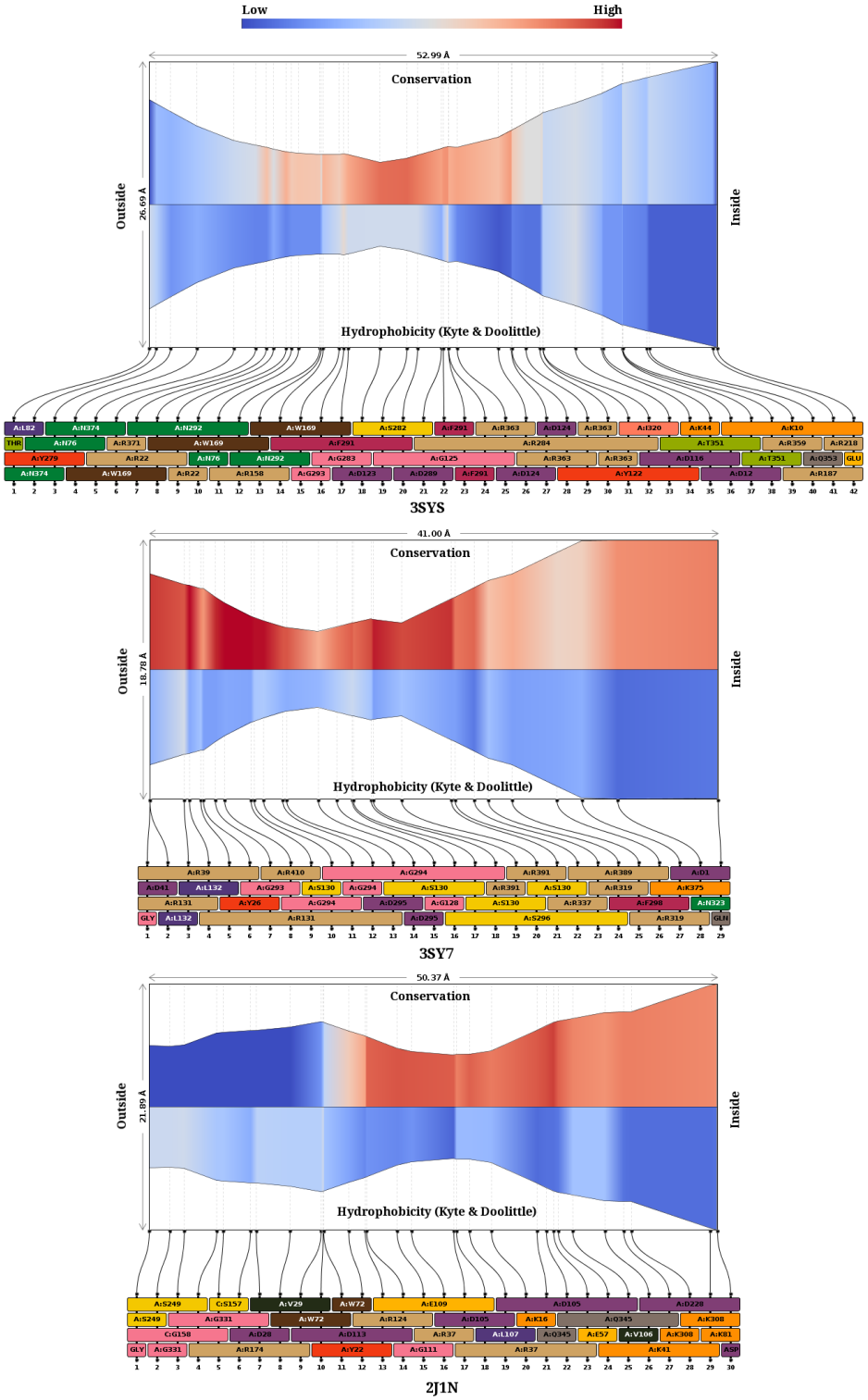


Figure 3.23: The 2D representations of channels from top to bottom correspond to 3SYS, 3SY7 and 2J1N respectively. The split-profiles are coloured by conservation and hydrophobicity of residues, red denoting higher values while blue denotes low values. The amino acids lining these channels are also shown using box representation. A high proportion of basic residue Arginine (coloured as light brown box) in the channel neighbourhood can be clearly observed.

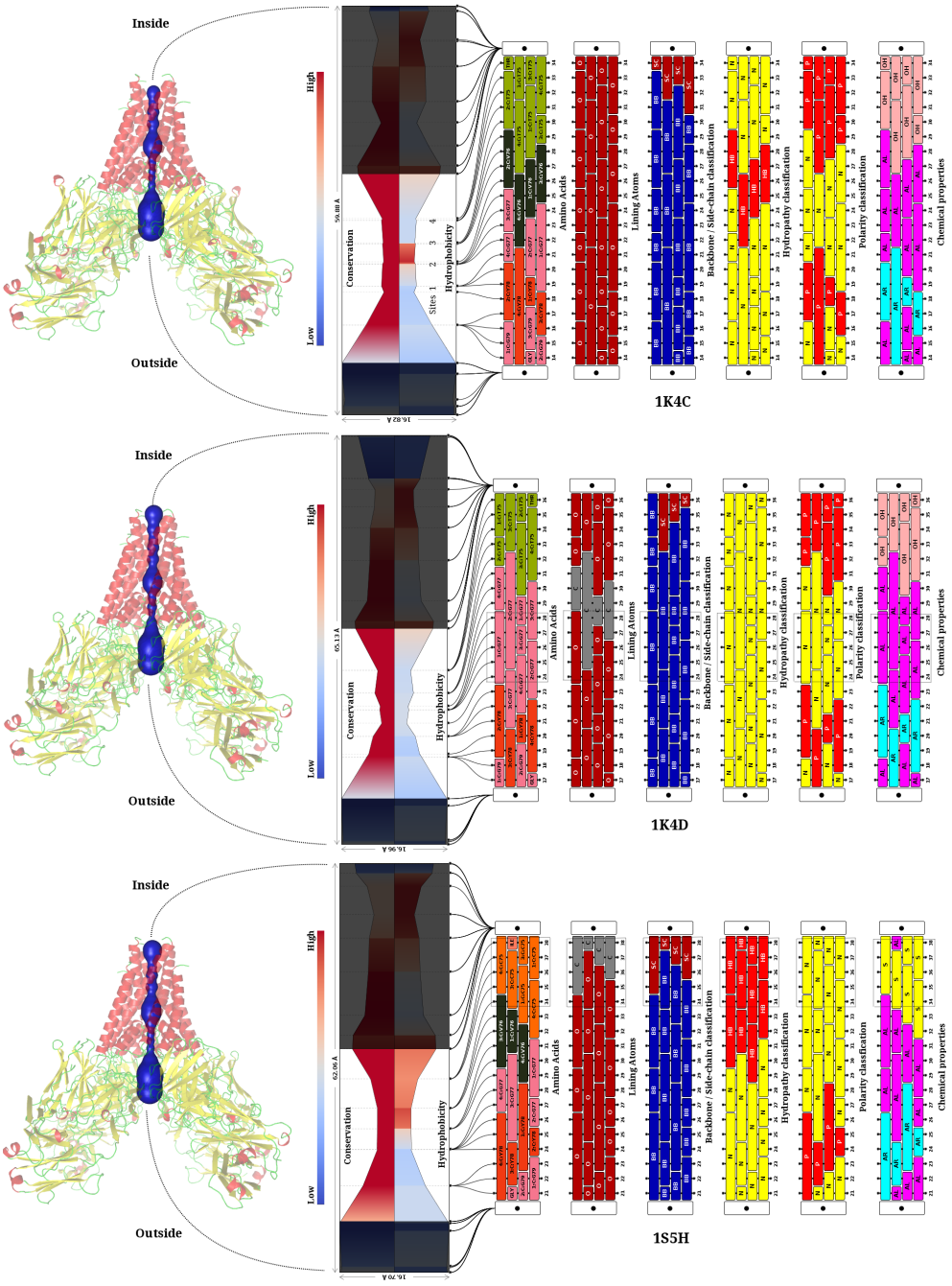


Figure 3.24: From top to bottom, properties of channels in 1K4C, 1K4D and mutant 1S5H are shown. The mutation of T75 to C75 is correctly captured by ChExVis as highlighted by a box in 1S5H lining residues. We can clearly observe a few green boxes change to orange in the mutated channel. This mutation also affects the physico-chemical properties of the channel as shown subsequent rows of the above figure. For example, chemical property, hydrophobicity and polarity exhibit clear change. Moreover, in the mutated structure there are fewer Oxygen lining the channel, which are critical for functionality of this transmembrane channel. Also, the channel becomes constricted in lower concentration of K^+ ions as captured in 1K4D channel profile. In this configuration, the critical Oxygen atoms are replaced by Carbon atoms as highlighted by a box.

Chapter 4

RobustCavities: Reliable Extraction and Exploration of Cavities in Proteins

As we described earlier in Chapter 2, empty spaces within the protein molecule are called *cavities*. Cavities with and without openings are referred to as *pockets* and *voids*, respectively. These structural features are known to play a key role in determining the stability and function of proteins [83]. Pockets often form part of the active site of enzymes or interacting sites for other proteins, while internal voids are often relevant structurally as features that affect the overall thermodynamic stability of the protein [62]. It is established that filling up internal voids improves the packing of the protein thus increasing stability [54]. In this respect, detecting and visualizing structurally robust cavities inside the protein informs the biologist on which mutations to perform to improve internal packing and get a stable protein. Given the importance of cavities in protein structure study, several algorithms and software are available to compute them given protein structure data, such as from the Protein Data Bank [7].

Protein structures are most commonly determined from X-ray crystallography data. Inaccuracies, noise, and more generally, uncertainty in the data adversely affects existing cavity detection methods. Small inaccuracies may already cause a difference in the reported number of cavities. The atom radii are also empirically determined and hence not precisely defined values. For example, as illustrated in Figure 4.1(a) and 4.1(b), presence of such inaccuracies may result in a cavity detection method reporting two distinct cavities instead of one, or report very small volume cavities. Faulty fragmentation and reporting of small cavities distract the focus to biologically irrelevant cavities. Figure 4.1(c) illustrates the problem as it occurs in a lyzosyme protein. To the best of our knowledge, uncertainty in the data and its effect on cavity computation is not considered by any of the available tools. Our goal is to develop an interactive method to compute *robust cavities* in

proteins, which enables the user to reduce, if not completely eliminate, the inaccuracies in cavity computation.

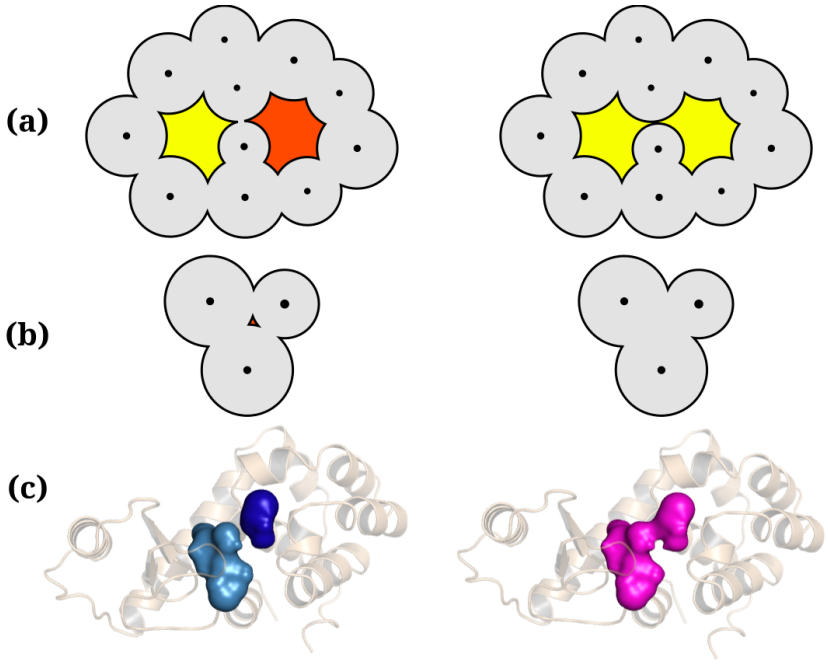


Figure 4.1: **(a)** Two cavities that are apparently very near to each other may be a single cavity. **(b)** A very small cavity may be reported whereas no such cavity may exist. **(c)** The two cavities that appear very near to each other in a lyzozyme protein (200L) may be a single cavity. The solid surface represents cavities while the protein is shown as cartoon to provide context.

Numerous methods have been proposed for identification of cavities in protein molecules. These methods employ a wide variety of approaches – grid and occupancy based, graph based, model fitting, Monte Carlo simulations on solvent molecules, and Voronoi diagram based. Early tools used grid-based approaches to extract cavities [49, 63]. These methods discretized the space occupied by the protein thereby trading off accuracy in favour of computational efficiency. The notion of cavities and their classification as voids and pockets was more formally defined using the alpha shape model of a molecule proposed by Edelsbrunner et al. [33, 34] and Liang et al. [65, 66]. This enabled accurate identification of cavities and further supported the computation of geometric properties like volumes and surface areas. Tools based on the above approach are available and widely used [30, 56]. Graph based methods have also been used to identify cavities and compute their volumes [78, 94]. Given an estimate of the empty space, Varadarajan et al. describe a Monte Carlo procedure to position water molecules within to improve the accuracy of the extracted cavity [17]. Several recent methods are based on the Voronoi diagram of the atoms [68, 74, 75, 80, 81]. Techniques for identification of cavities and channels are summarized very well in a recent review paper by Krone et al. [60].

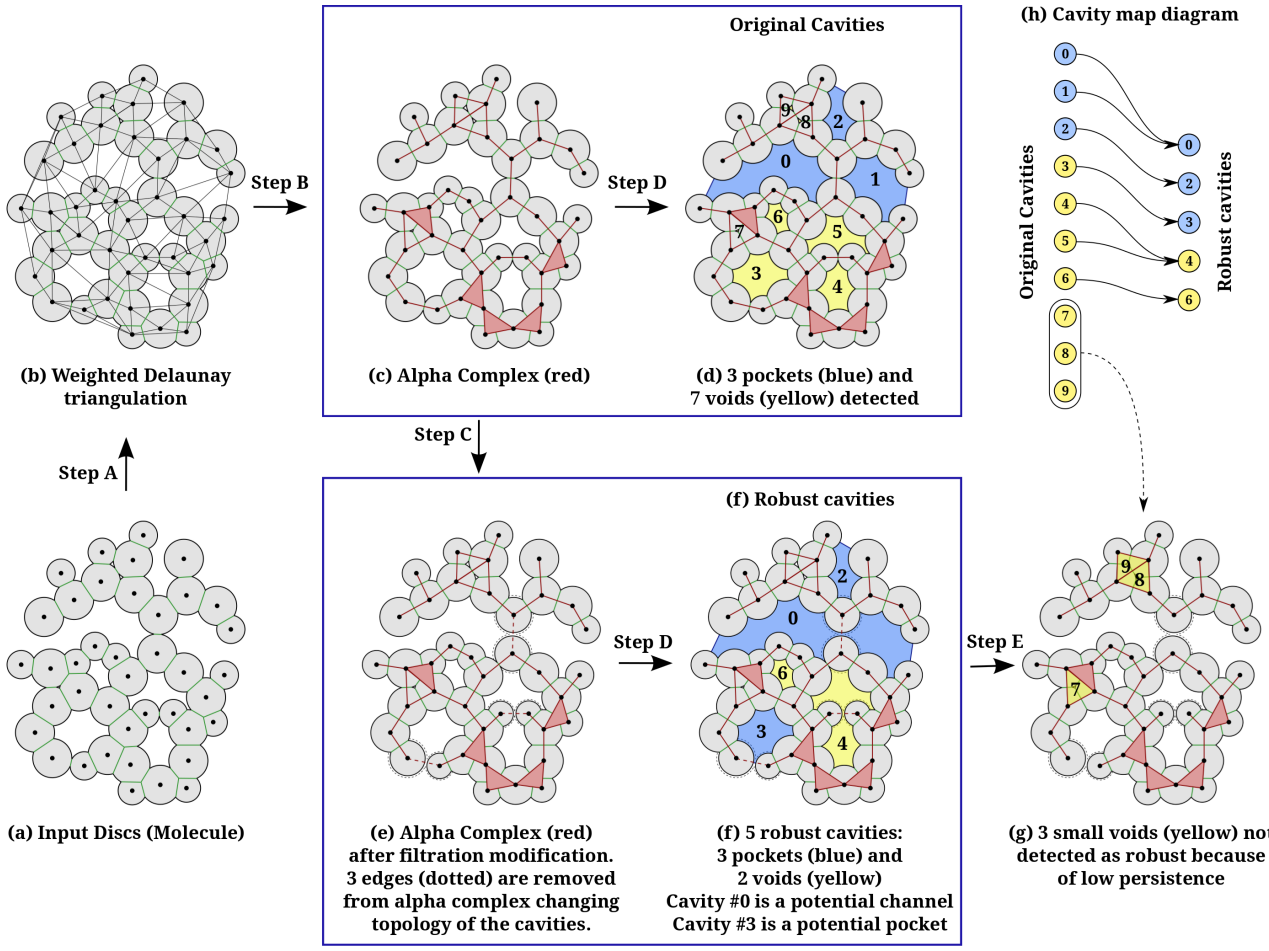


Figure 4.2: 2D illustration of cavities and robust cavities. **(a)** A molecule is represented as a set of discs (balls in 3D). **(b, c)** Weighted Delaunay triangulation and alpha complex are computed for the input discs. The alpha complex is always a subset of the Delaunay triangulation and its complement represents the empty space within the molecule. **(d)** The connected components within the empty space denote cavities. **(e)** For a given value of ε, we simulate the growing/shrinking of select atoms by modifying the alpha complex resulting in merging of cavities. **(f)** The set of *robust cavities* are found by detecting connected components in the modified complement space. **(g)** In addition, the parameter π is also used to prune out cavities with low persistence. **(h)** In the formation of robust cavities, the original cavities are either retained or merged or pruned. This mapping between original and robust cavities is captured using the *cavity map diagram*.

We use the notion of *robust cavities* proposed earlier by Sridharamurthy et al. [89]. These cavities are stable under small perturbations of atomic radii. We describe an efficient method for the computation of robust cavities, which builds upon the alpha shape-based framework of cavity extraction. This method is implemented and available as a web-server tool called RobustCavities[1].

[1] http://vgl.csa.iisc.ac.in/robustCavities/

The web-server, in addition to detecting robust cavities and calculating volumes, also provides an interactive framework that the biologist can use to explore and study important cavity structures within the molecule. The software also supports exporting the detected cavities with the relevant biochemical context to enable their visualization in PyMOL [28]. Finally, we use this software to demonstrate the applicability of the notion of robust cavities in detection of potential channels, pockets and other biologically relevant cavities in several proteins.

4.1 Robust cavities and their computation

We recommend that the reader refer to Chapter 2 for technical background on alpha complex based cavity representation. Also, detailed discussion on robust cavities, their properties and efficient computation, is also available in the methodology papers [89, 90]. The notion of robust cavities is 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. A cavity is called an ε-*stable* cavity if it remains a single connected cavity within all α-complexes for α values in the range $[-\varepsilon, \varepsilon]$. Topological persistence of a cavity is defined as the range of α values when the cavity is present in $D - K_\alpha$. A cavity is π-*persistent* if its topological persistence is greater than π. Combining the two notions of robustness, a cavity is called (ϵ, π)-*stable* if it is both ε-stable and π-persistent. The term *robust cavities* refers to cavities that are either stable as is or can be made stable via a small perturbation in atomic radii. In order to identify robust cavities, we describe a method which symbolically modifies the radii of a select set of atoms by systematically processing and modifying the alpha filtration.

Given the molecule as a set of balls B, and values for ϵ and π, the robust cavities can be computed in five steps:

Step A: Compute weighted Delaunay triangulation of B.

Step B: Compute alpha complex at $\alpha = 0$. The value of α can also be supplied as an input parameter.

Step C: Depending on value of ε, identify triangles and tetrahedra in the alpha complex which can be safely moved to the end of the filtration. This crucial step is called *filtration modification* and its consequence is that nearby cavities get merged into single cavity.

Step D: Identify connected components in the complement of the alpha complex to determine the cavities.

Step E: Robust cavities are pruned further based on persistence parameter π.

The above steps are illustrated using a 2D example in Figure 4.2. It should be noted that if steps

C and E are not executed, the method reduces to well known method for computing cavities using the alpha shape-based model. We also compute cavities without executing steps C and E; the set of cavities obtained in this fashion is referred to as the set of *original cavities*.

Cavity map diagram

The relationship between the sets of original and robust cavities is captured by a bipartite graph called *cavity map diagram*. Nodes in the bipartite graph are connected by an edge if the corresponding pair of original and robust cavities share common tetrahedra or triangles. Based on the definitions and the proposed method, it can easily be shown that any original cavity can be connected to at most one robust cavity. The cavity map diagram thus captures the merging behaviour of cavities in a succinct graphical representation, which can be leveraged for interactive exploration of cavities in the protein. By default, we color the node based on its classification as a void or pocket. However, the criteria can be varied to highlight different cavities in the diagram depending on user input. Figure 4.2(h) shows the cavity map diagram for the corresponding 2D example.

4.2 Results

4.2.1 Benchmarking and validation

To validate whether the method is able to correctly identify cavities, we created a set of 137 model mutants with known cavity locations. Given a protein, a model mutant is created by replacing a buried hydrophobic residue in the protein core with Alanine [92, 93]. We used 13 wild-type structures as base; they were mutated *in-silico* to provide 137 structures with artificial cavities. The replacement of a bulky residue with the small Alanine side chain results in the creation of an artificial cavity in the mutant (M) at a known location, which can be approximated as the centroid (p_M) of atoms removed from the bulky residue to form the mutant.

A cavity is computed as a set of connected tetrahedra. A cavity can also be represented as a set of orthogonal spheres where each sphere corresponds to a tetrahedron constituting the cavity. We can then define distance from a point q in space to the cavity as the minimum distance from q to the centres of spheres representing the cavity. Using this measure, we can find the distance to the closest cavity from any given point q.

For a given mutant (M), cavities are extracted using RobustCavities in both M and the wild-type (W) structure from which M is obtained. Since the residue which is mutated is completely buried in wild-type structure, we expect the distance d_W to the nearest cavity from the expected mutant cavity location (*i.e.* p_M) to be high in wild type structure. However, the mutation should create a cavity very close to p_M in M, and thus distance to nearest cavity d_M should be smaller.

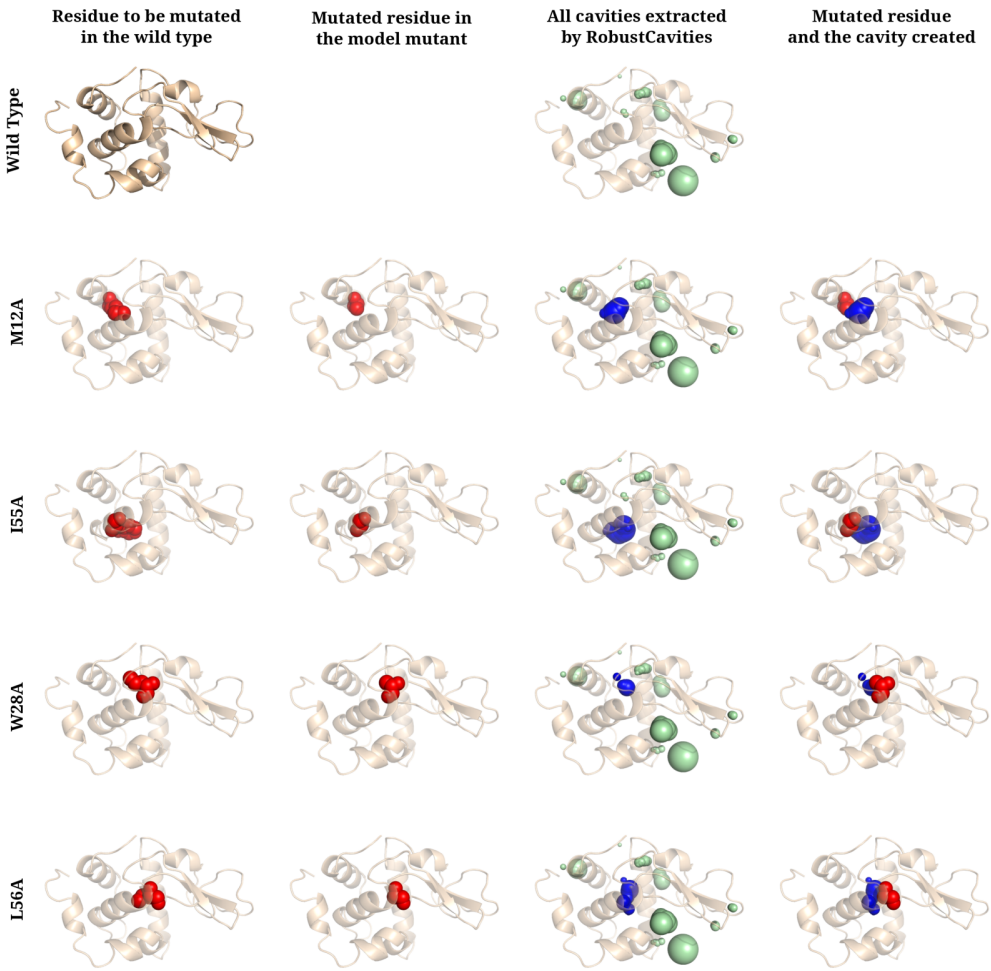


Figure 4.3: Demonstration of validation experiment with an example PDB structure and some of its mutants. In first row the wild type structure of 193L, and the cavities extracted by RobustCavities (light green) are shown. In the next four rows results for four model mutants are shown. The name of the mutant is mentioned on the left. A mutant is given name 'XyA' if the residue X in the wild type structure with id y is mutated to Aniline (A) in the mutant model. For example, M12A means the residue id 12 which was a Methionine is mutated to Alanine. In the first column, the original (not mutated) residue in the wild type structure is shown as red spheres. In the second column, the mutated residue i.e. Alanine which replaces the original residue is shown (red spheres) in the mutant model. Replacement of a large buried residue with small Alanine is expected to create a detectable cavity. The third column shows the cavities extracted by RobustCavities. The cavity which is distinct from the cavities extracted in wild-type is highlighted in blue. In the fourth column, we show the new cavity detected (blue spheres) in the mutant along with the Alanine residue (red spheres). Compare the images in the fourth column with the first and second columns, the blue spheres correspond exactly to the atoms removed to create the model mutant, thus confirming the ability of RobustCavities to detect these cavities.

So, if we find $d_M < d_W$, then it is indicative of successful detection of the artificial cavity in the mutant. Out of 137 mutants, we observed $d_M < d_W$ for 135 structures resulting in success rate of 98.5%. For the two remaining mutants, we observed $d_M = d_W$ indicating the artificial cavity in the mutant was not detected. These two structures had Valine to Alanine mutations. Valine is one of the smaller hydrophobic residues and its replacement would result in very small cavities, which is probably why RobustCavities failed to identify them. Refer to the Figure 4.3 for validation result for on of the protein in our dataset.

Validation results

The tables 4.1, 4.2 and 4.3 list detailed observations for the benchmarking experiment performed to validate the correctness of the implementation of RobustCavities for cavity detection by assessing its ability to detect cavities in artificially created mutants with known cavities. Please note that this experiment was not performed to observe the effect of the parameters ϵ and π. Experiments to observe the effect of the parameters ϵ and π were also performed and have been reported in papers by Sridharamurthy *et al.* [89, 90].

Thirteen PDB structures (193L, 1A19, 1BVI, 1FKD, 1HFX, 1IG5, 1LZ1, 1MBD, 1RUV, 1STN, 2CI2, 2EQL, 3DFR) were taken. From these wild type structures, 137 model mutants were created by replacing *in silico* a buried hydrophobic residue with Alanine. The mutant is given name 'XyA' if the residue X in the wild type structure with id y is mutated to Aniline (A). For example, M12A means the residue id 12 which was a Methionine is mutated to Alanine. The mutation should create a small cavity in the molecule, expected location (p) which can be estimated by computing the centroid of the set of atoms removed from the wild-type structure to create the mutant. d_W is the minimum distance from the expected cavity location p to the cavities detected in wild type structure. Similarly, d_M is the minimum distance from p to the cavities detected in the model mutant. If RobustCavities correctly identifies the artificial cavity then we can expect $d_M < d_W$. We observed this condition to be true in 135 out of the 137 cases. These are marked as '✓' in the last column. For two mutant d_M was found to be equal to d_W, which suggests the mutant cavity was not detected by RobustCavities. These cases are marked as 'X' in the Result column. Both these mutants are Valine to Aniline mutations, which results in very small cavities. The parameters used for this experiment were: $\epsilon = 0$, $\pi = 0.0$ and solvent radius = 1.2 Å.

Table 4.1: Validation results for RobustCavities on mutant models

PDB	Mutation	Cavity Centroid (p_M)			d_W	d_M	$d_W - d_M$	Result
		x	y	z				
193L	I55A	-0.165	16.870	14.121	3.257	0.994	2.263	✓
	I58A	1.473	17.755	23.746	3.203	0.956	2.247	✓
	L17A	-8.215	19.752	19.226	4.645	1.026	3.619	✓
	L56A	-1.478	20.335	18.297	3.190	1.036	2.154	✓
	L83A	2.115	13.394	23.086	3.206	0.808	2.398	✓
	L8A	-3.756	18.178	11.511	3.263	1.125	2.138	✓
	M105A	-4.602	27.080	20.113	3.141	0.479	2.662	✓
	M12A	-5.830	19.549	14.617	3.209	0.165	3.044	✓
	V29A	-5.627	23.076	10.326	7.779	1.858	5.921	✓
	W28A	-6.869	23.681	19.369	3.111	0.088	3.023	✓
1A19	F56A	84.320	54.227	6.977	3.133	0.283	2.850	✓
	F74A	91.081	58.676	0.029	3.137	0.489	2.648	✓
	I5A	93.053	53.627	5.871	3.169	1.626	1.543	✓
	I84A	90.471	56.976	-5.796	6.313	0.967	5.346	✓
	I86A	86.620	53.899	-2.211	3.202	1.017	2.185	✓
	L16A	91.149	58.640	8.905	3.13	1.031	2.099	✓
	L20A	94.392	58.595	4.271	3.177	1.256	1.921	✓
	L37A	92.715	62.816	1.647	3.246	1.018	2.228	✓
	L41A	92.573	61.618	-2.965	3.179	0.972	2.207	✓
	L49A	94.753	55.130	-4.049	3.189	1.161	2.028	✓
	L51A	91.392	53.994	0.822	4.137	1.363	2.774	✓
	L71A	84.495	56.142	2.196	3.202	1.010	2.192	✓
	L88A	83.849	51.315	2.370	3.202	1.093	2.109	✓
	W53A	88.668	54.303	6.075	3.060	0.398	2.662	✓
1BVI	F80A	10.221	22.486	44.244	3.086	0.055	3.031	✓
	I61A	14.749	15.56	52.257	3.383	0.869	2.514	✓
	I90A	21.187	22.229	47.819	3.197	0.878	2.319	✓
	V78A	13.781	18.105	48.484	3.331	1.865	1.466	✓
	V79A	17.504	22.934	44.581	5.607	1.883	3.724	✓
	W59A	8.671	21.948	51.361	3.145	0.198	2.947	✓
1FKD	I76A	11.865	33.789	6.502	5.687	0.967	4.720	✓
	L103A	13.976	26.026	19.810	3.626	1.012	2.614	✓
	L74A	10.124	30.648	11.558	5.899	0.965	4.934	✓
	L97A	12.715	31.102	0.795	3.198	0.807	2.391	✓
	M66A	9.854	28.452	16.549	3.064	0.576	2.488	✓
	V101A	12.052	26.174	12.154	3.226	1.971	1.255	✓
	V24A	16.139	24.248	14.311	3.226	1.835	1.391	✓
	V63A	14.613	29.735	15.448	3.226	1.777	1.449	✓
1IG5	F66A	6.604	9.232	25.794	3.085	0.231	2.854	✓
	L23A	6.562	14.673	26.945	3.389	0.835	2.554	✓
	L28A	11.117	13.149	24.807	3.158	1.053	2.105	✓
	L31A	8.786	11.034	29.908	5.236	0.908	4.328	✓
	L69A	11.609	10.541	20.991	3.158	0.935	2.223	✓
	V61A	6.905	14.911	20.495	3.155	1.783	1.372	✓

Table 4.2: Validation results for RobustCavities on mutant models (cont.)

PDB	Mutation	Cavity Centroid (p_M)			d_W	d_M	$d_W - d_M$	Result
		x	y	z				
1HFX	F53A	8.393	18.446	14.717	3.078	0.265	2.813	✓
	I29A	8.852	22.741	6.837	3.306	0.951	2.355	✓
	I30A	10.229	17.169	10.435	3.209	1.098	2.111	✓
	I55A	10.602	22.964	19.590	3.155	1.013	2.142	✓
	I95A	7.191	26.711	17.862	3.562	0.969	2.593	✓
	L12A	5.714	14.301	11.785	3.132	1.052	2.080	✓
	L15A	2.901	18.766	13.356	3.262	0.950	2.312	✓
	L27A	6.846	13.203	5.737	5.751	0.911	4.840	✓
	L52A	12.043	14.649	13.593	4.343	1.312	3.031	✓
	L80A	11.449	19.986	22.469	3.240	0.755	2.485	✓
	L8A	9.411	12.315	10.719	3.741	0.862	2.879	✓
	V92A	5.757	21.218	15.900	4.764	4.764	0	X
	W104A	8.380	24.780	12.925	3.097	0.231	2.866	✓
	W26A	5.112	20.651	10.218	3.083	0.096	2.987	✓
1LZ1	F57A	13.029	17.200	27.980	3.092	0.267	2.825	✓
	I56A	9.394	18.038	24.699	3.283	0.900	2.383	✓
	I59A	10.639	12.826	32.608	3.179	0.867	2.312	✓
	L12A	14.547	22.574	25.973	3.171	1.031	2.140	✓
	L31A	16.883	13.210	26.443	3.178	0.775	2.403	✓
	L84A	6.384	14.822	33.238	3.182	0.953	2.229	✓
	L8A	12.028	21.774	22.530	3.171	0.926	2.245	✓
	M17A	16.421	21.797	30.713	6.082	0.322	5.760	✓
	M29A	16.555	21.767	20.678	5.178	0.619	4.559	✓
	V100A	19.988	15.872	33.950	5.413	5.413	0	X
	W28A	19.002	18.846	29.388	6.530	0.303	6.227	✓
1MBD	F138A	9.838	22.890	6.155	3.106	0.690	2.416	✓
	F33A	21.523	22.480	-5.915	3.076	0.319	2.757	✓
	I111A	16.605	17.226	5.212	3.169	1.304	1.865	✓
	I142A	6.403	26.241	4.037	3.262	0.935	2.327	✓
	I28A	20.685	17.682	3.262	3.167	1.220	1.947	✓
	L115A	19.438	13.322	9.054	3.168	1.202	1.966	✓
	L135A	12.131	16.643	7.069	3.198	1.055	2.143	✓
	L29A	21.785	23.044	-0.496	3.156	0.839	2.317	✓
	L32A	17.023	19.784	-3.927	3.230	0.917	2.313	✓
	L61A	26.070	24.189	-3.368	3.140	0.985	2.155	✓
	L69A	21.161	21.119	6.935	3.169	0.797	2.372	✓
	L72A	14.681	21.252	8.415	3.195	1.202	1.993	✓
	L76A	13.443	19.440	12.402	3.198	1.303	1.895	✓
	M131A	14.456	14.032	10.555	3.126	0.787	2.339	✓
	V10A	12.576	14.718	14.546	3.201	1.895	1.306	✓
	V17A	22.062	17.152	10.976	3.168	1.891	1.277	✓
	W14A	18.731	19.316	12.167	3.066	0.241	2.825	✓
2CI2	I39A	6.458	-25.470	14.641	6.056	1.022	5.034	✓
	I76A	-0.405	-20.698	14.913	3.156	0.802	2.354	✓
	L68A	4.339	-20.415	14.989	3.945	0.911	3.034	✓
	V66A	8.318	-23.463	11.755	5.145	1.892	3.253	✓

Table 4.3: Validation results for RobustCavities on mutant models (cont.)

PDB	Mutation	Cavity Centroid (p_M)			d_W	d_M	$d_W - d_M$	Result
		x	y	z				
1RUV	F120A	32.498	18.138	12.221	2.418	0.205	2.213	✓
	F46A	24.410	15.146	15.709	5.370	0.290	5.080	✓
	F8A	28.150	23.088	10.290	2.403	0.075	2.328	✓
	I106A	34.755	21.283	16.471	6.149	0.400	5.749	✓
	I81A	32.742	16.114	19.525	2.185	0.487	1.698	✓
	M30A	22.507	9.927	12.164	2.220	0.272	1.948	✓
	V108A	34.415	23.673	11.397	5.937	1.232	4.705	✓
	V47A	29.603	20.434	17.619	6.838	1.123	5.715	✓
	V54A	30.183	25.435	16.497	2.604	1.276	1.328	✓
	V57A	35.776	26.838	15.401	6.144	1.321	4.823	✓
1STN	F34A	11.678	27.346	20.607	3.461	0.171	3.290	✓
	I92A	8.964	27.497	15.166	3.163	0.910	2.253	✓
	L103A	4.793	19.837	9.470	5.500	0.859	4.641	✓
	L125A	-2.874	24.668	13.736	6.716	1.274	5.442	✓
	L25A	14.060	27.598	16.024	3.168	0.984	2.184	✓
	L36A	6.541	22.675	14.447	3.166	1.235	1.931	✓
	V104A	-2.813	20.538	10.561	3.210	1.835	1.375	✓
	V23A	12.302	23.351	14.367	3.168	1.726	1.442	✓
	V39A	1.307	20.523	14.471	5.088	1.702	3.386	✓
	V66A	12.755	25.096	9.801	3.168	1.835	1.333	✓
	V74A	10.511	31.537	18.405	5.746	2.133	3.613	✓
	V99A	7.204	25.205	10.104	3.302	1.679	1.623	✓
2EQL	F57A	29.612	-3.201	18.891	3.103	0.360	2.743	✓
	L12A	30.897	-5.410	13.729	3.182	1.227	1.955	✓
	L56A	29.129	0.296	15.867	3.626	0.961	2.665	✓
	L59A	29.979	-1.938	24.682	3.157	1.031	2.126	✓
	L83A	33.266	2.071	24.537	3.187	0.983	2.204	✓
	L8A	27.731	-1.851	11.315	3.182	1.319	1.863	✓
	M17A	32.475	-8.884	17.148	6.659	0.392	6.267	✓
	M31A	24.515	-5.096	19.387	3.125	0.489	2.636	✓
	V29A	25.352	-5.782	10.463	5.865	2.057	3.808	✓
	W28A	28.163	-10.09	17.786	6.166	0.103	6.063	✓
3DFR	F103A	2.445	23.384	21.744	3.099	0.144	2.955	✓
	F3A	1.946	19.551	24.781	3.087	0.278	2.809	✓
	I96A	4.195	23.028	28.188	4.014	1.026	2.988	✓
	L113A	-3.814	16.668	23.861	3.249	0.894	2.355	✓
	L118A	-8.688	28.979	13.777	3.214	1.101	2.113	✓
	L4A	-4.181	27.567	25.848	3.182	0.934	2.248	✓
	M39A	1.931	32.149	28.820	3.744	0.616	3.128	✓
	V115A	-7.233	21.215	19.759	3.155	2.007	1.148	✓
	V40A	6.712	26.275	30.190	4.370	1.705	2.665	✓
	V41A	5.535	31.319	26.128	3.156	1.707	1.449	✓
	V60A	8.540	29.137	33.005	3.238	2.040	1.198	✓
	V61A	10.873	33.282	27.851	6.718	1.815	4.903	✓
	V82A	9.573	24.578	32.365	3.218	1.964	1.254	✓
	V95A	-1.606	26.107	30.465	3.160	1.896	1.264	✓
	W5A	-2.918	21.714	19.964	4.667	0.153	4.514	✓

4.2.2 Web-server workflow

RobustCavities accepts a protein structure in the PDB file format. The user can either provide a PDB ID or upload a file. The user also supplies values of the solvent radius and the two stability parameters (ε and π). The parameter ε is a local measure of stability, which determines if nearby cavities can be merged into a single cavity. Higher values mean select atomic radii can be perturbed by a greater amount resulting in more merged cavities. Thus higher ε may be used to model more flexible and packing deficient structures. However, we recommend that ε is kept below 1.53 which corresponds to a maximum radius perturbation of 0.5Å for any structure, as exceeding this value may result in spurious connections. The second parameter π corresponds to persistence, and is a global measure of the importance of the cavity. Higher value of π implies that a larger number of small cavities would be pruned out from the set of robust cavities. If $\pi = 0$, none of the cavities will be removed. It should be noted that persistence roughly corresponds to the volume of the cavity, which means removing low persistence cavities usually removes cavities with low volumes. Probe radius is another parameter which can be provided by the user. The default value corresponds to the solvent radius for water *i.e.* 1.4Å. We recommend it should not be modified for most cases. Refer to Figure 4.4 for snapshot of job submission form and more details about the input parameters.

For the supplied input, the web-server computes both the original and robust cavities, along with their properties such as volume, surface area, and chemical composition. The results for the given input are displayed within the browser window using JSMol. The result page offers a lot of flexibility in terms of interactive exploration of cavities. The cavities can be viewed as union of balls or as skin surface, while the protein can be displayed in space-fill, cartoon or wire-frame representations. The cavity map diagram is also displayed depicting the merging behaviour of the cavities.

In the default display mode (Figure 4.5), either all original or all robust cavities can be viewed at a time. The properties of the cavities are shown in a tabular form. A row in the table can be selected to highlight the corresponding cavity in the JSMol viewer. The web-server also supports a single cavity mode (Figure 4.6), in which the cavity map diagram becomes interactive. The nodes in the cavity map diagram can be highlighted based on different criteria. The user can then select a node in the diagram to view the corresponding cavity in detail. Optionally, the atoms lining the cavity can also be listed and highlighted in the viewer. The geometric properties and physico-chemical composition of the selected cavity is also displayed. Moreover, the cavities can be downloaded and explored off-line using PyMOL scripts generated by the server.

The RobustCavities web-server can be used for extraction and interactive exploration of robust cavities in any protein structure. The method works equally well for computing either completely

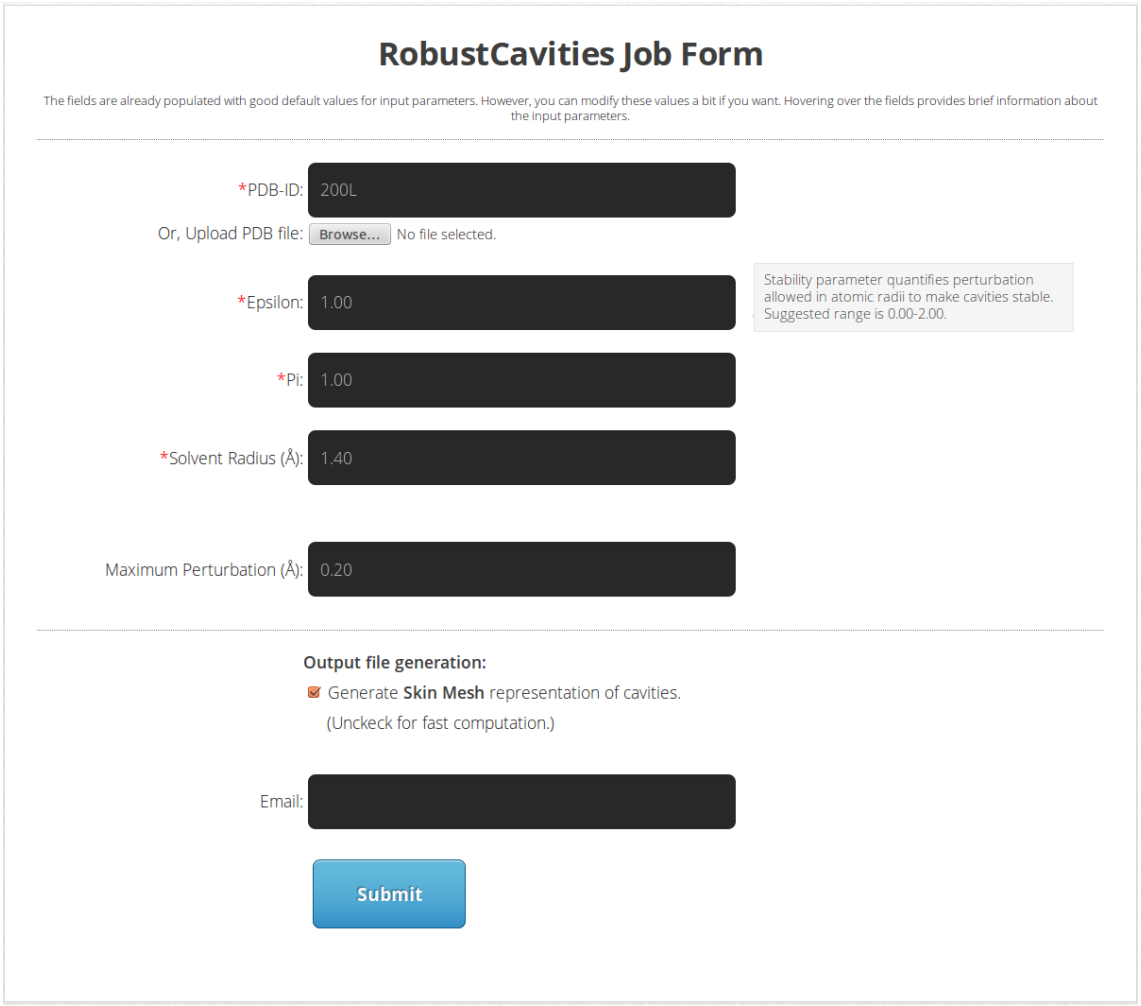


Figure 4.4: The Job submission form of RobustCavities server. Four inputs are essential without which the job cannot be submitted. Firstly, either PDB id should be specified or a PDB file in simple text format should be uploaded. Second, the stability parameters ϵ and π should also be specified. The default values work well in general, however they can be modified if required. Hovering over the fields gives succinct information about that parameter and the recommended range of values which should be supplied. The fourth essential parameter is the solvent radius. The default value used is 1.4 Å, which corresponds to the radius of water molecule. We recommend the value of solvent radius to be kept around the default value, as very low values may result in detection of single huge single cavity covering the whole empty space inside the protein. Lastly, there are two optional parameters. One is check box for generation of skin meshes for all cavities. This is a time consuming step but switched on by default, as it enhances user experience of exploration. This can be turned off to speed up the computation time significantly. Optionally, the user can also specify the Email ID. In such cases, the link to the job result page would be mailed after completion of the job. This is helpful if the job is time consuming, or there is long queue of submitted jobs on the web-server.

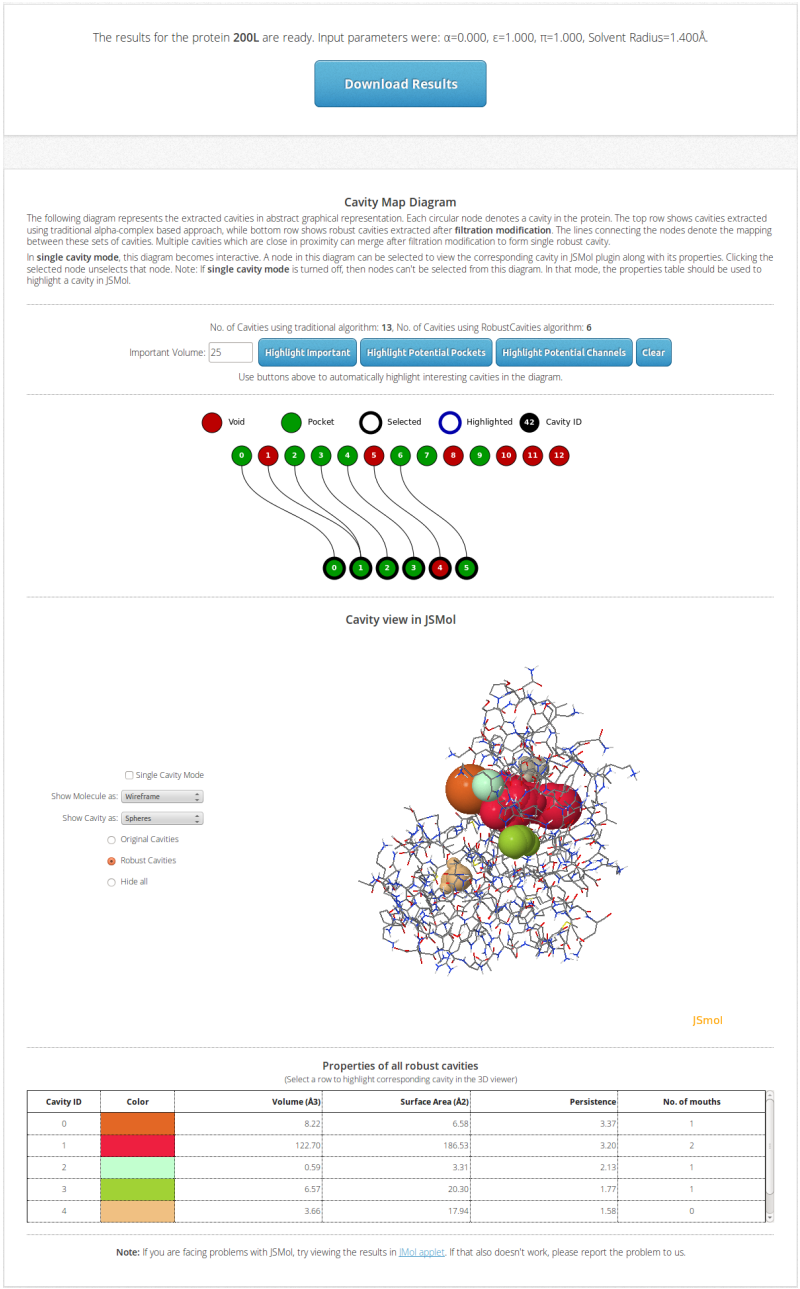


Figure 4.5: The default Results page generated for the protein 200L by the RobustCavities server. At the top, the download link for the generated results is provided. The zip file contains PyMol script for exploring the cavities offline. In the center of the page, the JSMol interactive applet is displayed. To its left is the control panel, through which the visual representations of the molecule and the cavities can be changed. The user can toggle between robust and original cavities. The properties of the selected cavities are shown in the properties table at bottom. The table is interactive, i.e. selecting a row highlights the corresponding cavity in JSMol.

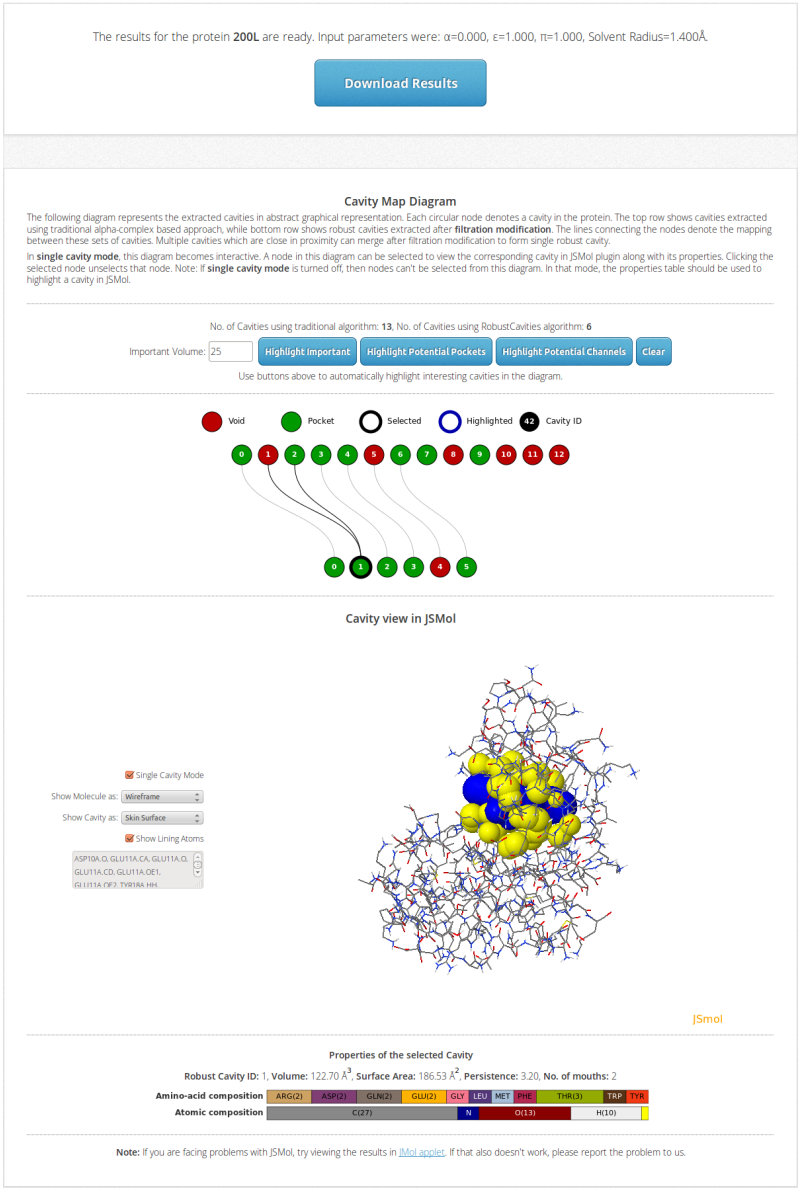


Figure 4.6: The Results page also supports a single cavity mode as shown above. This mode is activated by checking the 'Single Cavity Mode' check box, and deactivated to go back to default all cavities view by un-checking it. In this mode, only one cavity (coloured blue) is displayed at a time. Also, the cavity map diagram becomes active in this mode. A node can be selected in that diagram to highlight and explore the corresponding cavity in detail (here, robust cavity with id 1 is selected). The nodes in the cavity map diagram can be highlighted based on different criteria which is controlled by the button panel above the diagram. The geometric properties and chemical composition of the selected cavity is shown at the bottom. The atoms lining the cavity can be highlighted (coloured yellow) in the JSMol view and listed in the text area in the control panel.

buried voids or partially exposed pockets. A few pre-computed examples are available on the web-server. In the next section we describe a particular use case of the method *i.e.* detection of *potential channels*.

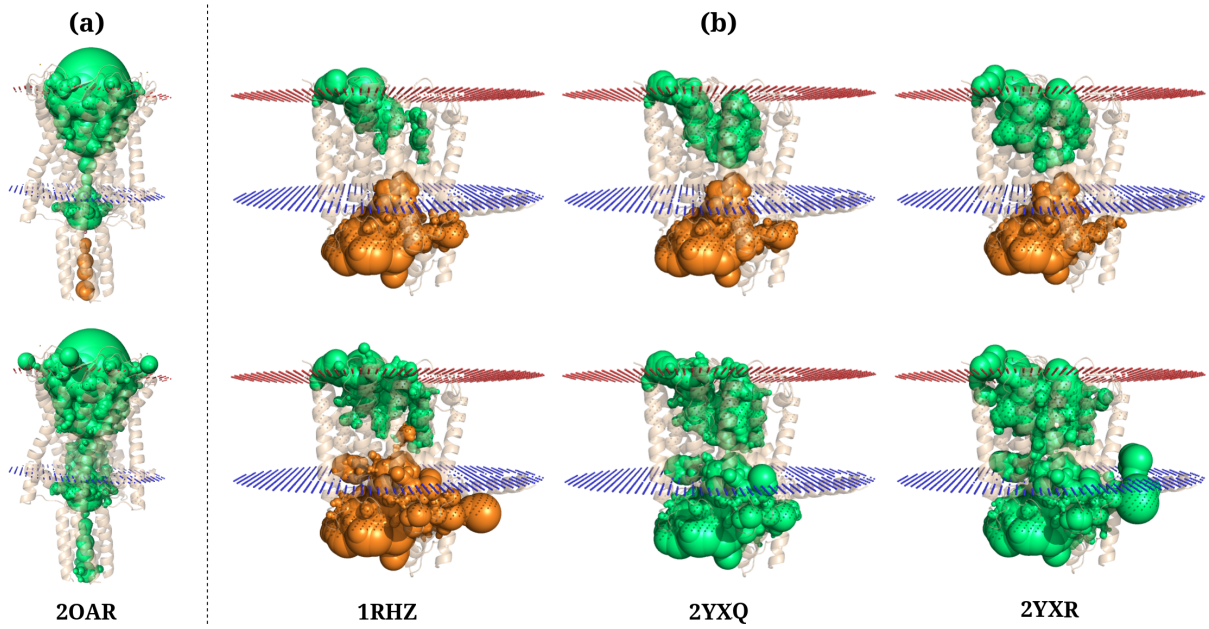


Figure 4.7: The demonstration of the capability of RobustCavities to detect potential channels. The top row shows the original cavities while the bottom row shows the robust cavities. The PDB ids are mentioned at the bottom. The protein is shown in cartoon representation, while red and blue dotted planes denote outer and inner layers of the cell membrane. **(a)** The known transmembrane channel in 2OAR is detected as fragmented in original cavities. The channel is successfully identified as a contiguous robust cavity in roust cavities. **(b)** The three columns show the result for different PDB structures of SecY transmembrane protein [97, 76]. The transmembrane channel is not detected in all three structures in original cavities. RobustCavities cavities successfully detects the transmembrane channel in the half-open mutants 2YXR and 2YXQ, although the same channel is not found in the closed wild type structure 1RHZ, which is known to be more tightly packed.

4.2.3 Detection of potential channels

Passage of solutes through the cell membrane is tightly regulated by membrane proteins by opening or closing their channels. In general, cavity detection programs are unable to detect a clear channel through closed or half open conformations of membrane proteins. Using RobustCavities with appropriate input parameter values and the cavity diagram interface, the channel can be detected as a single robust cavity. This is of value since few membrane protein structures are available and often structures for different conformational state for the same membrane protein are unavailable. A similar approach helps in finding *potential pockets* that may be incorrectly labelled as voids due to inaccuracies in the data. For example, Figure 4.7(a) shows a mechanosensitive channel for large

conductance (2OAR). While the original cavities get fragmented because of the small neck of the closed protein, the robust cavities (at $\varepsilon = 1.4$) show a clear channel through the protein. Similarly, Figure 4.7(b) shows the SecY membrane protein which forms an integral part of the translocon assembly [97]. In its closed state (1RHZ), the plug domain of the proteins maintains a tight seal and prevents any leakage [76]. In the closed state both the original cavities and robust cavities do not show a transmembrane channel. However, even in the two half-open states of the protein (2YXQ and 2YXR), the original cavities are unable to capture the channel while robust cavities successfully delineated the channel (at $\varepsilon = 1.4$).

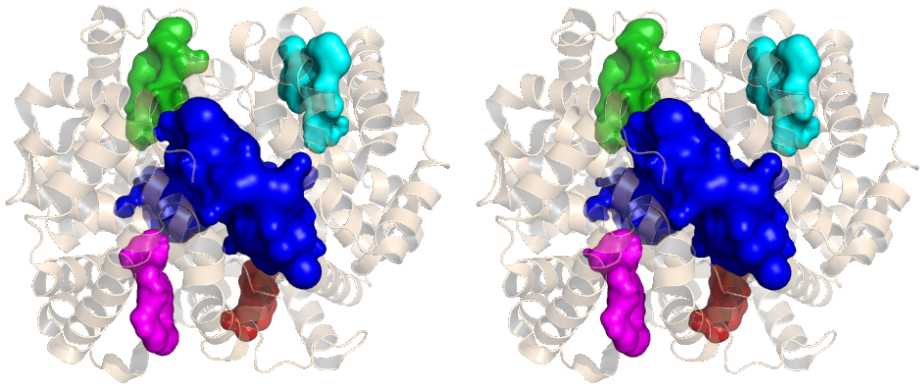
4.2.4 Robust cavities in Hemoglobin

We studied cavity structures in low and high affinity states of the protein Hemoglobin. As shown in Figures 4.8(a) and 4.8(c), both low and high affinity structures consist of four heme sites surrounding a central cavity. Also, all these cavities are disconnected at $\alpha = 0$ in both the structures. However, after modifying filtration, while the topology of cavities in low affinity structure remains unchanged (Figure 4.8(b)), two heme sites in chains B and D of high affinity structure merge with central cavity (Figure 4.8(d)). It is known that Oxygen binding to heme in hemoglobin causes a conformational change in the rest of the structure which leads to an increase in oxygen binding affinity. The binding results in the conformation transition from tense form (low affinity T state) to relaxed form (high affinity R state). This important conformational change is being correctly captured by the change in topology of the cavities of the R state (Figure 4.8(d)).

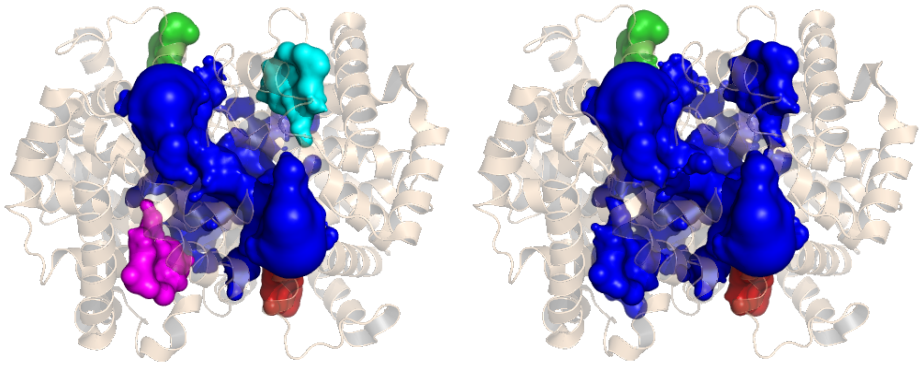
4.2.5 Robust cavities in Myoglobin

Myoglobin functions as an oxygen storage and delivery protein in the heart and skeletal muscles. The gas molecule ligand binds to the Fe atom present in the heme moiety buried deep within the protein [13]. The Fe-ligand bond is photosensitive and many studies have used this property to probe Myoglobin dynamics [21, 14]. Further, Myoglobin binds xenon (Xe) where Xe populates four pre-existing cavities in Myoglobin [95].

We computed original and robust cavities in wild-type sperm whale myoglobin (1DUK) bound to heme and Fe. As shown in Figure 4.9, the original cavities contain the four separate Xe binding sites (Xe^1 to Xe^4) along with the distal pocket (DP) where ligand carbon monoxide (CO) binds to Fe, apart from many other smaller cavities. It has been shown in previous structural studies that upon dissociation due to photolysis, CO occupies the Xe^4 [14] and Xe^1 [21] cavities for the maximum amount of time before exhibiting geminate binding or escape from protein matrix into solvent. Both these sites (Xe^1 and Xe^4) are successfully detected by the original cavities. Extended molecular dynamics simulation have shown that the path that CO takes to migrate from the distal



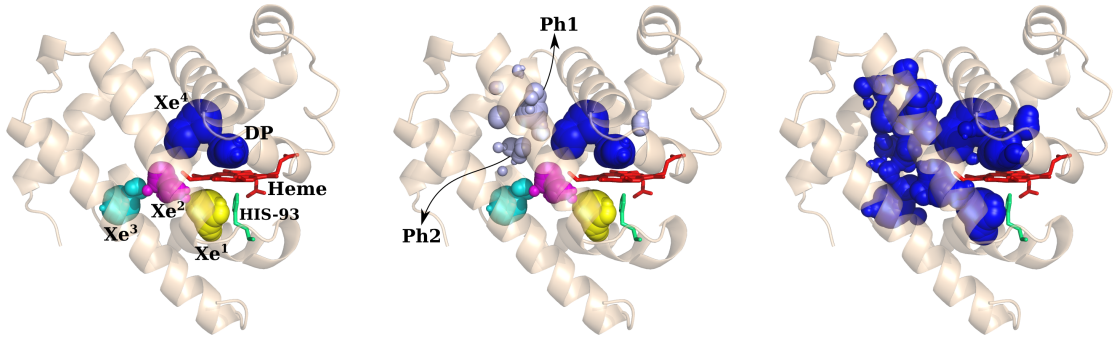
(a) 1HGA - Original (b) 1HGA - RobustCavities



(c) 1BBB - Original (d) 1BBB - RobustCavities

Figure 4.8: Cavity structures in two states of Hemoglobin. In all the images, central cavity is shown in blue color, while red, cyan, green and magenta colors are used for heme sites in chains A, B, C and D, respectively. The first row in the figure is for 1HGA which is low affinity T state, while second row corresponds to high affinity R state (1BBB) of Hemoglobin. (a) The central cavity and four heme cavities in low affinity state of Hemoglobin. (b) Even after applying modification, the heme sites do not merge with central cavity. (c) The central cavity and four heme cavities in high affinity state of Hemoglobin. (d) The heme sites in chains B and D merge with the central cavity after application of RobustCavities.

side of heme to proximal side traverses through the alternate Xe binding site [10]. We extracted robust cavities at $\varepsilon = 1.15$ which corresponds to a maximum radius perturbation of 0.27Å. As shown in Figure 4.9(c), the robust cavities detects a single cavity corresponding to xenon binding sites, DP and two other cavities, thus revealing a connection from the distal cavity to the proximal side of heme. RobustCavities suggests direct connections between Xe^1 and Xe^2, Xe^2 and Xe^3, and between Xe^3 and Xe^4 via other smaller cavities. Thus, proximal Xe^1 is connected with Distal Pocket tracing a direct path through xenon binding sites around the heme moiety. This result agrees completely with the results of the extended molecular dynamics simulation which have previously been performed [10].



(a) (b) (c)

Figure 4.9: Extraction of robust cavities in Myoglobin. Myoglobin is shown in cartoon representation while heme moiety is shown in red using stick representation. The residue His 93 is also shown in green using stick representation for context. The region above heme is the distal side while the region below is the proximal side. (a) In original cavities, the five relevant binding sites appear disconnected. Only DP and Xe^4 (blue) are connected. (b) The original cavities which include the significant sites and a few other cavities in the vicinity (light-blue). No path from DP to Xe^1 is found. (c) RobustCavities with $\varepsilon = 1.15$ reports a single robust cavity connecting the cavities shown in (b). Direct connections between sites Xe^1 and Xe^2, and sites Xe^2 and Xe^3 are found. Xe^4 is found to be indirectly connected to the sites Xe^2 and Xe^3 via small intermediate cavities including Ph1 and Ph2.

4.3 Discussion and conclusion

RobustCavities is the first attempt in the direction of cavity extraction from uncertain data. We demonstrated that RobustCavities was able to pull out biologically relevant channels when run on a set of membrane proteins, despite the protein being present in a closed conformation. In comparison, CASTp [30] which also uses alpha shape-based framework, fails to identify these channels. When compared to CASTp, RobustCavities allows radii of select atoms to be grown or shrunk, and therefore supports a search over space of molecular structures to compute relevant cavities. Hence, this method provides a flexible framework for computing cavities from uncertain input data.

The utility of accounting for small changes in atomic radii can be fairly judged when studying a packing deficient protein such as Myoglobin. Running RobustCavities on the wild type sperm whale Myoglobin structure with default parameters, we successfully report the two major cavities which the cognate ligand CO resides in during the geminate binding phase of dissociation. Movement of the CO from its actual binding site, the distal pocket (DP) towards the Xe^4 site is facilitated by mutation of a single Leucine residue to Tyrosine [14]. This small positional change is implicitly captured by the method to report a biologically significant cavity. However the method with default

parameters is still not able to connect the distal and proximal binding sites of CO. This is because the conformational change required for the CO to move from distal to proximal side is more than what can be accounted for by default parameters. When we increase the value of ε appropriately (from 1.0 to 1.15), the method is able to capture the conformational change and reports a channel which connects DP to Xe^1 via Xe^4, Xe^3 and Xe^2. In an extended molecular dynamics simulation study by C. Bossa *et. al.* on wild type sperm whale myoglobin it was observed that over the time-scale of 80 ns, transient cavities form and collapse due to protein dynamics [10]. Two cavities which the authors labelled as Phantom 1 (Ph1) and Phantom 2 (Ph2), highlighted in Fig 4.9(b) were deemed important. Whereas Ph1 seemed a stable cavity existing for 98% of time, Ph2 was a transient cavity occurring 33.5% of the time over the duration of the simulation. These cavities played a crucial role in movement of CO from DP to Xe^1 since they connected the spatially distant Xe^4 and Xe^3 sites in Myoglobin. Indeed it was found that during the course of its journey CO resides in Ph2 for 0.1 ns and inhabits Ph1 for as long as 3.2 ns. Our results, obtained by running the program on a static structure, agree completely with the results of the molecular dynamics simulation. As seen in Fig 4.9(c), the robust cavities delineate a path connecting DP to Xe^4, Ph1, Ph2, Xe^3, Xe^2 and then to Xe^1 (in that order). Despite close spatial proximity of Xe^4 and Xe^2 the program does not directly connect the two and instead chooses a more circuitous path to connect cavities, which is consistent with the results of the extended molecular dynamics simulation. This aptly showcases the ability of the method to extract meaningful connections between cavities. An important caveat while using the method is to keep the value of ε within reasonable limits. Increasing its magnitude in a bid to account for more uncertainty will lead to irregular results.

The RobustCavities web-server interface provides significant improvements over existing tools in terms of analysis and visualization of cavities. Highly interactive exploration of cavities, supported by the novel cavity map diagram interface, within the browser is a feature not offered by most tools. Secondly, the web-server supports multiple representations of cavities. In addition to the volumes and surface areas of the cavities, chemical composition of the cavity is also computed and displayed, which allows study of physico-chemical properties of the cavities. In the single cavity mode, cavity lining residues can also be displayed leading to quick identification of cavity or residue of interest. Positions of atoms in a protein may have different degrees of uncertainty based on the structural feature (loops, helices, *etc.*) that they are part of, and based on their location (core, near the surface) within the protein. Similarly, uncertainty in atomic radii may vary based on atom type. RobustCavities currently does not consider these aspects and assumes the degree of uncertainty to be same across the molecule. In future, it would be interesting to consider use of different uncertainty parameters for different regions in a protein. Computation of robust cavities and channels in molecular dynamics data at interactive rates is another challenging problem worth exploring.

Chapter 5

An Integrated Geometric and Topological Approach to Connecting Cavities in Biomolecules

Study of cavities and channels in molecular structure is a crucial step in understanding the function of biomolecules. Current tools and techniques for extracting these structural features are sensitive to uncertainties in atomic position and radii. In this chapter, we study the problem of cavity extraction in biomolecules while taking into account such uncertainties. We propose an approach that connects user-specified cavities by computing an optimal conduit within the region occupied by the molecule. The conduit is computed using a topological representation of the occupied and empty regions and is guaranteed to satisfy well defined geometric optimality criteria. Visualization of the set of all cavities with multiple linked views serves as a useful interface for interactive extraction of stable cavities. We demonstrate the utility of the proposed method in successfully identifying biologically significant pathways between molecular cavities using several case studies.

5.1 Introduction

In Chapter 4, we discussed how uncertainty in the data adversely affects existing cavity detection methods (See Figure 5.1). In general, the existing methods for extraction of cavities do not explicitly handle the adverse effects of uncertainty in the data. Some methods support user-specified parameters such as solvent radius or a growth factor but they are almost always global parameters that affect all extracted cavities. To the best of our knowledge, the two-parameter solution we described in Chapter 4 for extraction of *robust cavities* is the first attempt in the direction of addressing the problem of uncertainty explicitly. However, that solution is also global in the sense that the param-

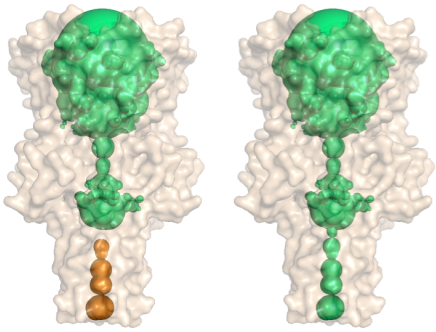


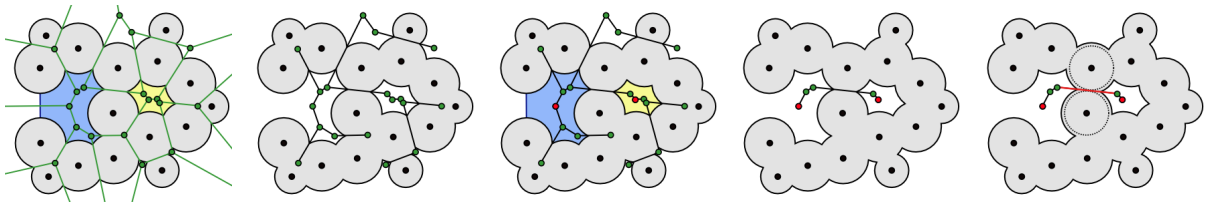
Figure 5.1: **Left:** The transmembrane channel through the protein 2OAR is detected as disconnected cavities. The default parameters are used to compute the cavities. **Right:** However, it can be connected by perturbing a few atoms around its bottleneck.

eters affect all extracted cavities. Such a solution may produce undesirable results by connecting cavities that lie outside the region of interest.

In this Chapter, we propose a simple and direct approach to address the problem, where the user examines the cavities and identifies artifacts or undesirable disconnections. The user interacts with the multiple linked views provided by the visualization and specifies a pair of cavities to be connected. Our cavity connection algorithm efficiently and automatically computes an optimal conduit between the cavities. Key contributions of this work include:

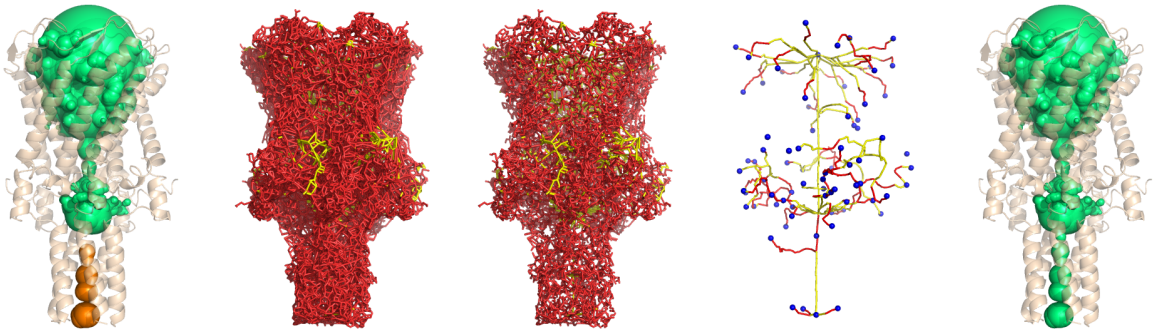
- A simple, explicit, and flexible method for extracting cavities in biomolecules from uncertain data with guaranteed bounds on the perturbation required.
- Efficient algorithms to compute a conduit between user selected cavities that satisfies well defined optimality criteria.
- Interactive visualization of cavities in a molecule with multiple linked views that facilitates identification of disconnected cavities.
- Three case studies that demonstrate the benefits of the cavity connection based method – computing ion transport channels from uncertain data, comparing cavities obtained from various mutants of a protein, and computing the migration path of carbon monoxide in myoglobin.

We evaluate the method by comparing the results with those obtained using a global parameter-driven cavity extraction method described in Chapter 4. We also note that our method may be used in conjunction with any of the Voronoi diagram based method [68, 80, 81] to improve the results.



(a) (b) (c) (d) (e)

Figure 5.2: Illustration of cavity connection method based on BOTTLENECK criterion using a 2D example. (a) The two cavities which are required to be connected are shown in the context of the molecule shown as a set of grey disks. The G_{MR} is shown in green. (b) The maximum spanning tree ($MaxST$) is computed for the network. (c) The representative nodes of the two cavities in the $MaxST$ are coloured red. (d) The connecting path detected between these cavities. (e) The only edge of the path which belongs to OR is highlighted in red. The lining atoms of this edge can be perturbed to physically connect these cavities.



(a) (b) (c) (d) (e)

Figure 5.3: Demonstration of cavity connection method applied to the protein 2OAR. (a) The two cavities which are required to be connected are shown in the context of the molecule shown in cartoon representation. (b) The complete dual graph G_{MR}. The edges which belong to OR are coloured red while edges belonging to ER are coloured yellow. (c) The $MaxST$ computed for G_{MR}. Same colouring scheme is used to identify edges in OR and ER. (d) The $MaxST$ is further pruned by restricting to paths connecting the cavity representatives. Here blue spheres show the cavity representatives. (e) Using cavity connection algorithm, the best path connecting the representative nodes of the two cavities shown in (a) is computed. The atoms are perturbed appropriately to obtain the merged cavity shown in this figure.

5.2 Cavity connection

Before continuing further, we recommend that the reader refer to Chapter 2 for the mathematical background required for this section. Cavities in a biomolecule, as defined in Chapter 2, are clearly sensitive to perturbations in atomic positions and radii. For example, Figure 5.1 illustrates a scenario

where perturbing a few atoms results in detection of a single large cavity instead of disconnected cavities. Recognizing the existence of such a single connected cavity and extracting it by performing the required perturbation is an interesting and challenging problem. Current approaches to cavity extraction employ global parameters to address this problem resulting in undesired merging of multiple cavities. We aim to develop a flexible user-driven method that can improve the results of the cavity extraction algorithm by supporting the automatic computation of an optimal conduit between two given cavities.

5.2.1 Problem statement

Given two disjoint cavities, the *cavity connection* problem is the computation of an optimal conduit between the cavities that (a) lies within the molecular region and (b) together with the two input cavities forms a single connected cavity after suitable perturbation of the atoms. We consider three optimality criteria that lead to different algorithms for connecting the cavities.

- BOTTLENECK : This is a min-max criterion where the objective is to minimize the maximum perturbation on an atom that will result in the formation of a conduit between the cavities.
- PROXIMITY : The objective here is to minimize the number of atoms perturbed to form the conduit.
- BOTTLENECK_PROXIMITY : This is a hybrid of the above criteria. The number of perturbed atoms is minimized given an upper bound on the maximum perturbation allowed on an atom.

5.2.2 Cavity connection methods

We now describe efficient algorithms to connect cavities satisfying each of the above-mentioned optimality criteria.

BOTTLENECK **criterion**

The uncertainty in atom locations and radii determined from x-ray crystallography maps motivate the development of methods that perturb the values to extract connected cavities. The BOTTLENECK criterion aims to limit this perturbation in the atom radii to the least possible value.

Consider the dual graph G_{MR} of the tetrahedra and triangles in the *Molecular Region* (MR). Nodes of this dual graph correspond to the tetrahedra and the arcs correspond to the triangles. A weight is associated with each arc of G_{MR}, equal to the smallest value of α at which the corresponding triangle is inserted into the filtration. Let C_i and C_j be the two cavities that the user would like to connect. Let t_i be a representative tetrahedron belonging to the cavity C_i and t_j be

the representative of C_j. Let n_i and n_j be the nodes in G_{MR} corresponding to t_i and t_j, respectively. The conduit between C_i and C_j may be represented by an alternating sequence of triangles and tetrahedra in MR and hence by a path in G_{MR}. In particular, we are interested in the path between n_i and n_j where the minimum weight over all arcs is maximized. We design a simple and efficient algorithm for computing this optimal path by recognizing that the path always lies within the maximum spanning tree of G_{MR}.

CLAIM. *The maximum spanning tree $MaxST$ of the weighted graph G_{MR} contains a path satisfying the* BOTTLENECK *criterion for all pairs of cavities.*

Proof. Consider two nodes n_i and n_j in G_{MR}. Let P_{ij} denote an optimal path between the two nodes and a_{ij} denote the minimum weight arc within the path. We describe a proof by contradiction. Let P'_{ij} ($\neq P_{ij}$) denote the unique path between n_i and n_j in $MaxST$ and a'_{ij} denote the minimum weight arc within the path. If the weights of a'_{ij} and a_{ij} are equal then P'_{ij} is also an optimal path. If the weight of a'_{ij} is smaller than a_{ij} then we can show that a new tree may be constructed with weight greater than $MaxST$ resulting in a contradiction. Delete the arc a'_{ij} from $MaxST$. This results in two disconnected trees containing the nodes n_i and n_j respectively. Let $b_{ij} \in P_{ij}$ be an arc connecting the partitions. Clearly, the weight of b_{ij} is greater than or equal to the weight of a_{ij} and hence the weight of a'_{ij}. Replace a'_{ij} with b_{ij} in $MaxST$ to obtain a spanning tree with greater weight than $MaxST$, a contradiction. Hence, $MaxST$ always contains an optimal path. □

The conduit may be computed from the optimal path P_{ij} by perturbing the atoms lining P_{ij}. The triangles in MR that are dual to arcs in P_{ij} belong to the *Occupied Region* (OR) and *Empty Region* (ER). We perturb the atoms incident on triangles in OR. Their radii are reduced by a value corresponding to the α-value at which the triangle is inserted into the filtration, thus establishing the connection between the cavities. Figures 5.2 and 5.3 illustrate this technique in 2D and 3D, respectively. A connection may also be established by perturbing the position of the atoms lining the path. However, computing such a perturbation without introducing steric clashes is a non-trivial and challenging task.

PROXIMITY **criterion**

Atoms that lie within the interior of the molecule are subject to greater physical constraints when compared to those lining the surface and hence less suitable for perturbation. The PROXIMITY criterion limits the number of atoms that are perturbed and hence limits the number of perturbed interior atoms.

Consider the dual graph G_{MR} as described earlier. Assign unit weight to the arcs that belong to OR and zero weight to arcs in ER. Let n_i and n_j be the representative nodes of cavities C_i and C_j

in G_{MR}, respectively. The optimal conduit between C_i and C_j is represented by the shortest path P_{ij} between nodes n_i and n_j in the weighted graph G_{MR}.

The conduit corresponding to P_{ij} may be computed by selecting one atom per arc in the path and shrinking it by a value corresponding to the α-value at which the triangle is inserted into the filtration. The number of atoms perturbed is thus equal to the length of the path contained in OR.

BOTTLENECK_PROXIMITY **criterion**

The BOTTLENECK_PROXIMITY is a hybrid of both criteria described above. Again, the optimal conduit is represented as a path. Given cavities C_i and C_j, we first compute the path P_{ij} satisfying the BOTTLENECK criterion. Let α_{min} be the weight of the minimum weight arc in the optimal path P_{ij}. Construct a sub graph G of G_{MR} induced by arcs whose weight is greater than α_{min}. Now, construct an optimal path P'_{ij} in G satisfying the PROXIMITY criterion. Alternatively, α_{min} may also be specified by the user instead of computing it using the BOTTLENECK criterion.

The conduit may be computed from P'_{ij} by selecting one atom per arc similar to the PROXIMITY criterion. However, the reduction in atom radius is now limited by α_{min}.

5.3 Visualization and interaction

We describe three linked interactive visualizations to help the user identify important cavities and connect them based on different criteria. Figure 5.4 shows these three views.

3D Visualization

In this view, the cavities are shown in the context of the molecule. The cavities can be shown as union-of-balls, where each tetrahedron in the cavity is represented by its power ball whose centre is equidistant from the four atoms and radius is equal to the power distance. Alternately, we can also display the cavity in its dual graph representation, where nodes are drawn at the centre of the power ball for each tetrahedron, and edges between the nodes correspond to the common triangle face. Each cavity is given a unique color which is consistently used across different visualizations to help identify the cavity quickly. The user can pick multiple cavities for connection by simply clicking on the 3D view of the cavity. The detected connecting paths are shown as a set of cylinders in the 3D view. The $MaxST$ and the pruned $MaxST$ which connects the cavity representatives can also be visualized in this view.

2D Visualization

This view shows the abstract representation of the cavities and their connections based on BOTTLENECK criterion. We construct a pruned sub-graph of the $MaxST$ containing only the edges and nodes needed for connecting the representative nodes of all the cavities. The graph is further

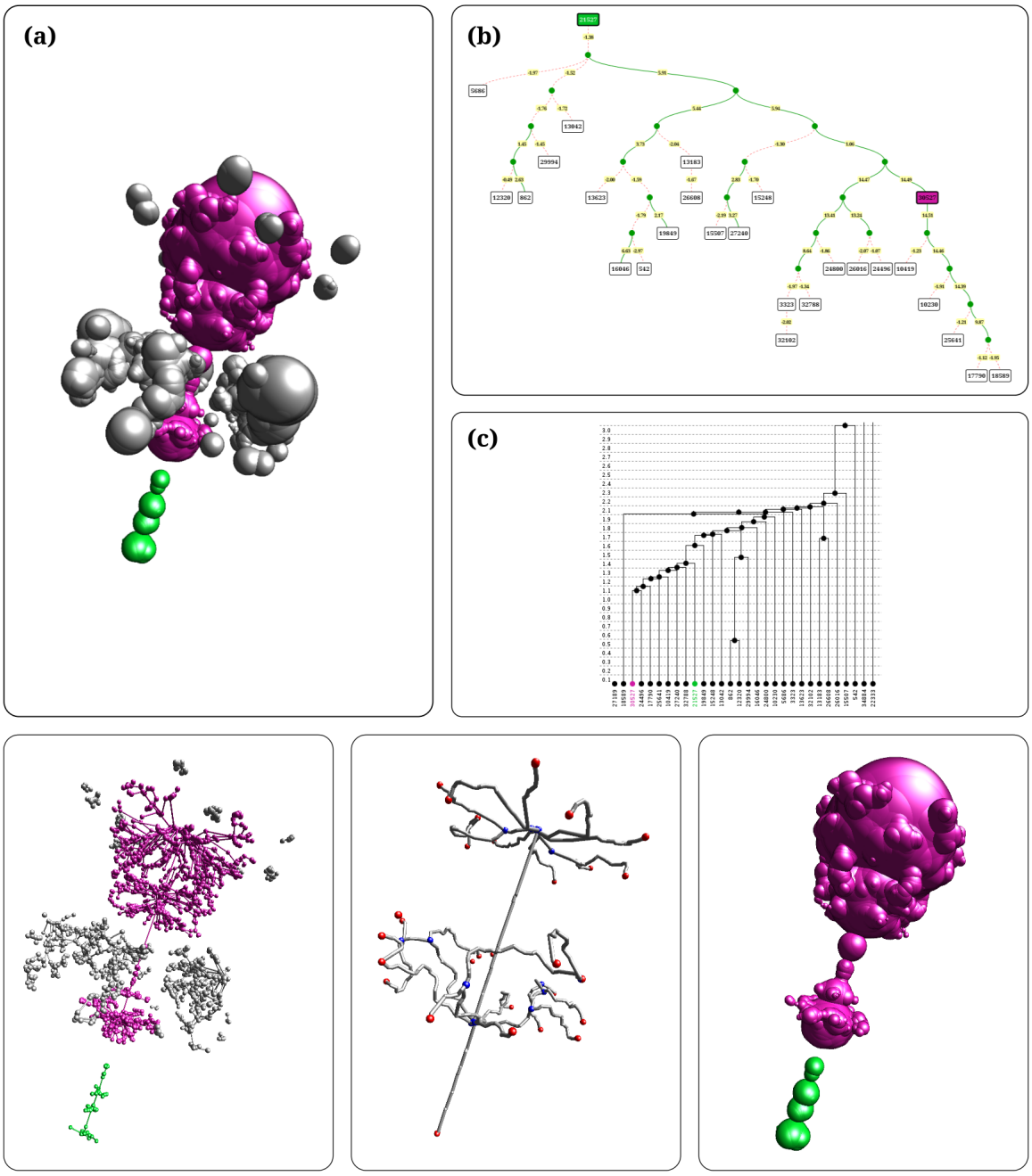


Figure 5.4: The three linked views of cavities in 2OAR are shown. Cavities may be selected from any of these views. **(a)** The 3D view shows the two cavities selected for connection in green and violet colors. Other cavities are shown in grey. **(b)** 2D graph visualization of the cavities. **(c)** This panel shows the cavity dendrogram in which the height is proportional to the α_{min} of the connecting path between cavities. Some additional 3D views are shown in the bottom row. From left to right: the dual graph representation, the simplified $MaxST$, and the two cavities to be connected.

minimized by collapsing paths into edges. After pruning and collapsing, the graph contains the representative nodes of all the cavities and a few connecting nodes. These nodes are connected by edges, each of which represents a path in the original $MaxST$. The nodes can be coloured and labelled based on different criteria. The edges are labelled by the minimum value of α in the corresponding path. This visualization is interactive and linked to other visualizations. The user can pick different cavities by selecting nodes in the graph. The connecting path is shown by highlighting the nodes and edges in the graph.

Hierarchical dendrogram

The negation of α_{min} of the optimal path P_{ij} connecting the cavities C_i and C_j can be treated as *cavity distance* measure. It can be shown cavity distance measure satisfies the non-negativity, coincidence, symmetry and triangle inequality properties. Based on this distance measure, we can cluster the cavities using hierarchical clustering and obtain the hierarchical dendrogram. This diagram shows the proximity of cavities based on BOTTLENECK criterion. It is a useful representation for showing the cavity hierarchy and may be used to identify cavities to connect. However, we do not use clustering to obtain this diagram, and instead construct this simultaneously during computation of $MaxST$.

User interaction

In addition to multiple interactive views of the cavities and their connections, the user is provided with other tools to identify important cavities. One such example is persistence based pruning of cavities. The user can interactively specify a threshold persistence value. The views are immediately updated and the cavities having less persistence than the specified threshold are removed from the three views. Another tool is automatic connection of all cavities using a perturbation below a given threshold. This is similar to what was proposed in [89].

5.4 Results and discussion

In this section, we first briefly discuss the implementation and runtime results. Then, we describe a qualitative comparison of our method with existing approaches for connecting cavities. Lastly, we demonstrate the utility of cavity connection using three example case studies. We show that our method can be used for connecting fragmented cavities to form channels. We can also study the propensity of a pathway being open based on the value of α_{min}. Using Myoglobin case study, we show that our cavity connection method can be used to reach similar conclusions as those reached after extensive molecular dynamics simulations. These case studies were carried out in collaboration with molecular biologists who are studying protein cavities and their effect on the stability of proteins. For all these examples, we use $\alpha = 0$, solvent radius of 1.4Å and BOTTLENECK criterion

for cavity connection unless specified otherwise.

5.4.1 Runtime results

Cavity connection method is implemented as a standalone interactive software in Java 1.6 and OpenGL 3.2. The following experiments were performed on a workstation with 8 core Intel Xeon processor and 16GB of RAM. The program requires weighted Delaunay complex, alpha complex and Delaunay flow as input. We assume these are already available. Using this as input, we compute the cavities, cavity representatives, cavity attributes such as persistence, G_{MR}, $MaxST$, and pruned $MaxST$ in a preprocessing step. The preprocessing times for the five molecules we discuss in this chapter are provided in Table 5.1. It should be noted that compared to α-complex computation time which takes a few seconds, the preprocessing time is significantly low. After preprocessing, the GUI is set-up and user can choose cavities for connection based on different criteria. The cavity connection time was observed to be in the range 2ms to 20ms for these molecules. This ensures that cavity connection can be done interactively.

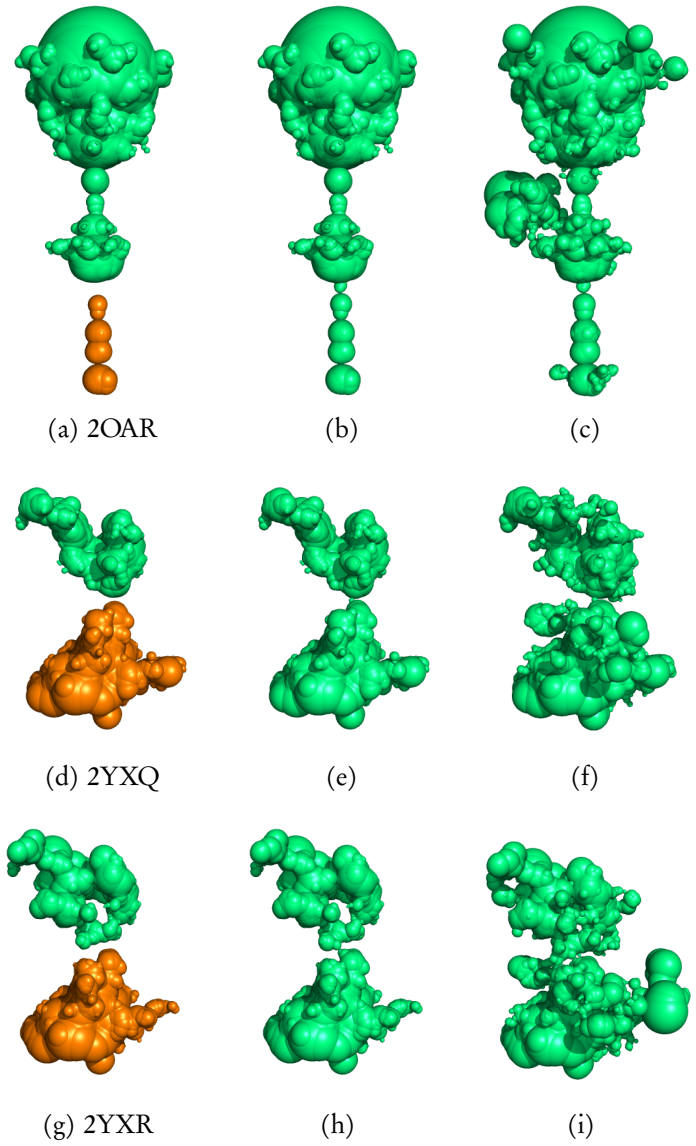
Table 5.1: Preprocessing times for different molecules in the study.

PDB id	#Atoms	Preprocess (sec)
2OAR	5772	0.357
1RHZ	4901	0.336
2YXQ	4853	0.337
2YXR	4789	0.298
1DUK	1256	0.111

5.4.2 Comparison

We perform qualitative comparison of our results with RobustCavities method proposed by Sridharamurthy *et al.* [89] and discussed earlier in Chapter 4. RobustCavities also attempts to remove inconsistencies in cavity detection by merging cavities into stable cavities using a global parameter ϵ. It is claimed that RobustCavities ensures that minimal change is done to the cavity volume by carefully modifying the atomic radii only for atoms lining the split triangles.

Figure 5.5 shows the result of our method and the RobustCavities for three protein structures. The cavity connection method causes limited perturbation to the atoms along the edges of the detected path (only edges in OR are modified). On the other hand, RobustCavities ends up connecting multiple cavities and significantly changes the volume of the merged cavity. The method proposed here provides more flexibility and finer control over cavity connection, and does very little change to the cavity volume, a desired outcome.



(a) 2OAR (b) (c)

(d) 2YXQ (e) (f)

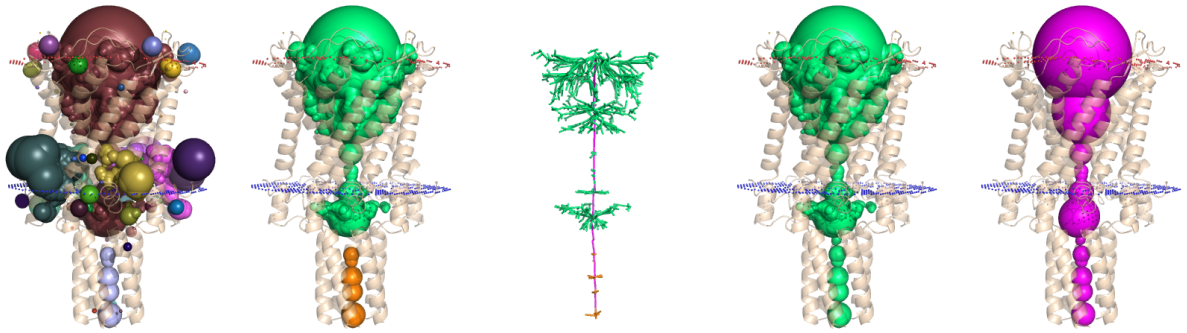
(g) 2YXR (h) (i)

Figure 5.5: Comparison of cavity connection results with RobustCavities. (a) The disconnected channel detected as two separate cavities (coloured green and orange) in 2OAR. (b) The cavity connection result. (c) The RobustCavities result. Clearly, the volume of the merged cavity has increased by a significant amount as compared to the result obtained by our cavity connection method. (d)–(f) Similar result is obtained for the protein 2YXQ. (g)–(i) The result obtained for the protein 2YXR.

It should be noted that other cavity and channel detection methods have user-defined parameters like solvent radius which can in principle be used to connect cavities. But they are global in nature, and significantly affect the cavity volume. Change in volume induced by RobustCavities is less than that induced by changing the solvent radius and extracting the cavities. Since, our method is

performing better than RobustCavities, we expect similar results when compared against other methods.

5.4.3 Mechanosensitive Channel of Large Conductance: Identifying a channel



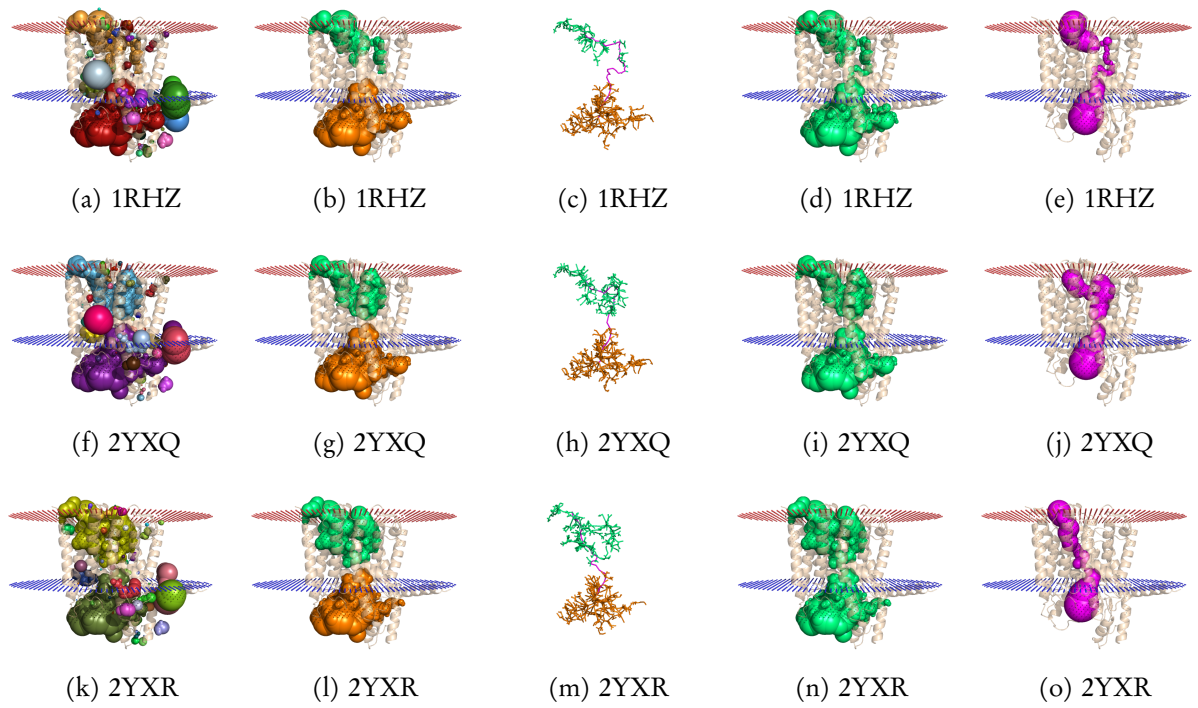
(a) (b) (c) (d) (e)

Figure 5.6: Cavity connection results for MscL transmembrane protein (PDB id: 2OAR). (a) All cavities detected in this protein are shown in the context of the molecule. The membrane is shown as red and blue layers. (b) We select two cavities at either end of the membrane for connection. (c) The connecting path found by the method is shown in pink. The two cavities shown in the dual graph representation for context. The maximum perturbation for the connecting path was found to be 0.4Å. (d) Single connected cavity after atom perturbation. (e) The connecting path helps identify the known ion transfer channel.

This molecule has been used as a running example in this chapter (Figures 5.1 and 5.3). MscL (PDB id: 2OAR) is a transmembrane ion transport channel. The transmembrane channel is detected as fragmented set of cavities instead of a single connected channel. The user selects two cavities at opposite ends of the channel (Figure 5.6(b)) and uses cavity connection method to find a good connecting path to merge these cavities. The α_{min} for the connecting path was found to be -1.38, which corresponds to maximum atomic perturbation of 0.4Å. Refer to Figure 5.6 for detailed results.

5.4.4 Translocase SecY: Comparing mutants

Translocase SecY is a transmembrane transporter protein which forms an integral part of the translocon assembly [97]. In its wild type closed state (PDB id: 1RHZ), the plug domain of the proteins maintains a seal and prevents any leakage [76]. Half and full plug deletion mutants of this protein were created to study this protein (PDB ids: 2YXQ and 2YXR, respectively). Even after plug deletion, these mutants attain packed structures and the channel is detected as a set of fragmented cavities. However, experimental data shows that plug deletions lead to increased rate of translocation of proteins and small molecules.



(a) 1RHZ (b) 1RHZ (c) 1RHZ (d) 1RHZ (e) 1RHZ

(f) 2YXQ (g) 2YXQ (h) 2YXQ (i) 2YXQ (j) 2YXQ

(k) 2YXR (l) 2YXR (m) 2YXR (n) 2YXR (o) 2YXR

Figure 5.7: The results for Translocase SecY case study. (a) The cavities detected in the wild type protein (1RHZ). (b) The cavities selected for connection. (c) The connecting path (pink) between these cavities. The maximum perturbation for this connecting path was found to be 0.69Å. (d) The resulting cavity after perturbation of atoms. (e) The connecting path as a channel across the membrane. (f)–(j) Similar results for the half plug deletion mutant (2YXQ) of the protein. The maximum perturbation for the connecting path was found to be 0.42Å. (k)–(o) The results for the full plug deletion mutant (2YXR) of the protein. The maximum perturbation was found to be 0.43Å.

We applied the BOTTLENECK criterion to connect the cavities along the channel on all the three structures. The detailed results are shown in Figure 5.7. The α_{min} of connecting paths were found to be -1.76, -1.41 and -1.43 for 1RHZ, 2YXQ and 2YXR, respectively. These values correspond to maximum atomic perturbations of 0.69Å, 0.42Å and 0.43Å, respectively. This clearly indicates that it is easier to open the channel in the mutants as compared to the wild-type, which supports the experimental evidence that the mutants are more conducive to transport of molecules through the channel.

5.4.5 Myoglobin: Identifying the migration path

Myoglobin (PDB id: 1DUK) functions as an oxygen storage and delivery protein in the heart and skeletal muscles. The gas molecule binds to the Fe atom present in the heme moiety buried within the protein [13]. This primary binding site where ligand carbon monoxide (CO) binds to Fe is

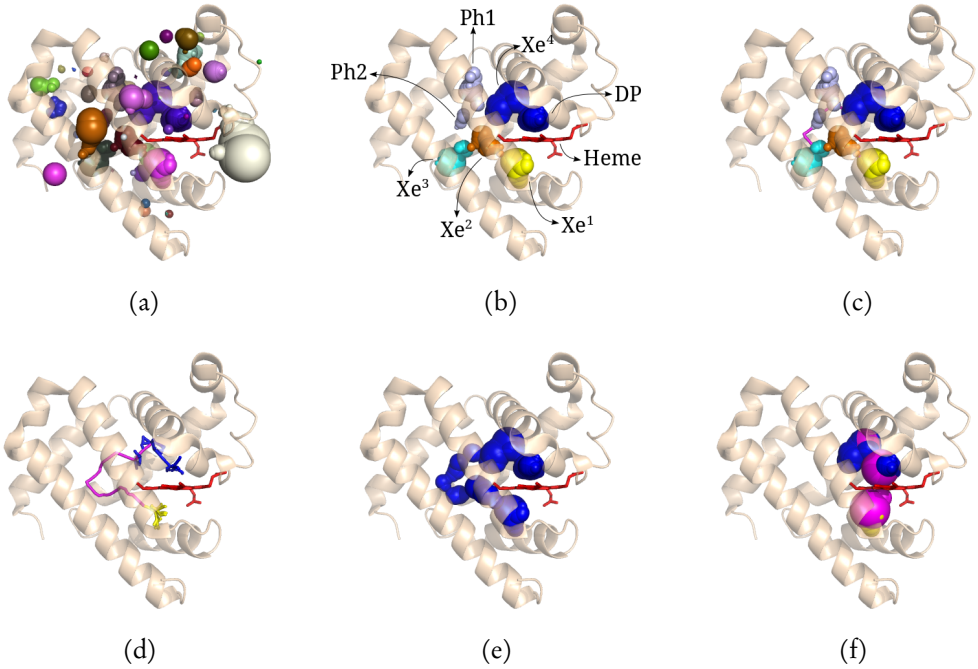


Figure 5.8: The results for Myoglobin case study. (a) The set of cavities detected in Myoglobin bound with heme. (b) The cavities of interest in this protein that have been studied earlier are labelled. We are interested in finding the connecting path between DP and Xe^1. (c) A connecting path (pink) from Xe^1 is detected that traverses through Xe^2, Xe^3, Ph2, Ph1 and Xe^4 to reach DP. This connection was suggested after extensive molecular dynamics simulations. However, we are able to detect this connection directly using the cavity connection method. The maximum perturbation required for detecting the path is found to be only 0.25Å. (d) The detected path (pink) along with dual graph representations of the two selected cavities. (e) The merged cavity (blue) formed after atom perturbation. (f) Using PROXIMITY criterion for finding the connecting path between DP and Xe^1 results in detection of direct path (pink) which does not pass through other Xe sites. The maximum perturbation for this path was found to be 2.16Å which suggests that direct connection between DP and Xe^1 is highly improbable.

referred to as the distal pocket (DP). Interaction of Myoglobin and Xenon (Xe) has been studied earlier and it was observed that Xe populates four pre-existing cavities in Myoglobin referred to as Xe^1 to Xe^4 [95]. Further, it has been shown in previous molecular dynamics simulation studies that CO occupies the cavities Xe^1 [21] and Xe^4 [13] for the maximum amount of time. Xe^4 is close to the distal pocket while Xe^1 is on the proximal side of the heme. The path taken by CO to migrate from distal side of heme to the proximal side and vice versa is of crucial significance for understanding the functionality of this protein.

In an extended molecular dynamics simulation study by C. Bossa *et al.* on wild type sperm whale myoglobin, it was observed that over the time-scale of 80 ns, transient cavities form and

collapse due to protein dynamics [10]. Two cavities which the authors labelled as Phantom 1 (Ph1) and Phantom 2 (Ph2), highlighted in Figure 5.8(b), were deemed important. Whereas Ph1 seemed a stable cavity existing for 98% of time, Ph2 was a transient cavity occurring 33.5% of the time over the duration of the simulation. These cavities played a crucial role in movement of CO from DP to Xe^1 since they connected the spatially distant Xe^4 and Xe^3 sites in Myoglobin. It was found that during the course of its journey CO resides in Ph2 for 0.1 ns and inhabits Ph1 for as long as 3.2 ns.

We applied BOTTLENECK criterion of cavity connection to find the connecting path between DP and Xe^1. We found that the connecting path passes through Xe^4, Ph1, Ph2, Xe^3 and Xe^2 to reach Xe^1. The α_{min} for the connecting path was found to be -1.11, which corresponds to maximum atomic perturbation of 0.25Å. This connection is shown in Figures 5.8(c), 5.8(d) and 5.8(e). Thus, we found that the proximal Xe^1 is connected with the distal pocket tracing a direct path through xenon binding sites around the heme moiety. This result agrees completely with the results of the extended molecular dynamics simulation performed earlier [10].

We also applied PROXIMITY criterion to find connecting path between DP and Xe^1. A short direct path with only two edges in OR was obtained, as shown in Figure 5.8(f). But, α_{min} for this connection was found to be -2.56 which corresponds to maximum perturbation of 2.16Å. This clearly suggests that direct connection between distal and proximal sides of the heme is impossible, or highly improbable. Hence, the path obtained by applying BOTTLENECK criterion is biologically significant.

5.5 Conclusions

Cavity detection methods suffer from unstable behaviour due to uncertain nature of protein structure data. We have described a novel solution via connecting molecular cavities under different optimization criteria. We described efficient solutions using an α-complex based internal representation of the cavities and the region occupied by the molecule. The computed connection helps in quantifying the 'connection distance' between cavities. This connection distance signifies the stability of the cavity in the presence of uncertainty. A larger distance implies increased difficulty to connect the cavities. Hence, they are expected to be more stable in uncertain dynamic environments experienced by the protein. An interactive visual interface with linked views aids the user in identifying interesting cavities to connect. There is scope to improve these visualizations and the user experience further. It is important to address the problem of channel and cavity extraction in uncertain data based on sound theoretical foundations. The methods proposed in this chapter can be adapted for other Voronoi diagram based methods like CAVER [81], MOLE [80] and the techniques proposed by Lindow et al. [68]. We believe that a user-driven flexible cavity connection capability would be a useful addition to these established channel and cavity detection tools.

Chapter 6

Facet-JFA: Faster Computation of Discrete Voronoi Diagrams

Voronoi diagram is one of the most widely used tools in computational geometry, with applications in computer graphics, image processing, mesh processing, robot navigation, and for data analysis in several scientific and engineering disciplines. A Voronoi diagram partitions space into regions given a set of seed points, with each point in a particular region being closer to its corresponding seed than to any other seed. The computation of Voronoi diagrams is one of the best studied problems in computational geometry, with optimal algorithms known for computing Voronoi diagrams on the plane [4, 5].

Discrete Voronoi diagram requires the computation of regions on a discrete grid of pixels, with seed points being some of the pixels themselves. Due to the inherent parallelism, this problem has been approached using the GPU. The first attempts to solve this problem using the GPU were done due to Hoff *et al.* [52]. Rong *et al.* [84] then used Jump Flooding, a parallel extension to flood-fill algorithm, which propagates label information from a pixel to the entire grid, to construct approximate Voronoi diagram. The GPU is harnessed to enable all the labelled pixels to propagate their label information, instead of a wave-front like approach. It can be easily shown that this approach requires $\log n$ steps to flood an $n \times n$ grid of pixels. Jump Flooding can be used to generate discrete Voronoi diagrams and the resulting algorithm is referred to as JFA (Jump Flooding Algorithm). Assuming fixed grid of size $n \times n$ with seed points already placed in the grid, JFA computes the Voronoi diagram in $\log n$ steps, and is thus independent of the number of seeds. The discrete nature of the problem introduces errors, which have been extensively documented in [84]. The algorithm has been implemented by Rong *et al.* using textures and pixel shaders and is one of the most efficient methods for computing the discrete Voronoi diagram till date. Cao *et al.* [16] proved that the

dual of the discrete Voronoi diagram constructed by flooding approach gives a geometrically and topologically correct dual triangulation.

In this chapter, we introduce a variant of JFA, called FACET-JFA, wherein only the pixels which are located near the Voronoi region boundaries are processed, thus immensely reducing the total amount of work done by the algorithm. The proposed approach first determines the lowest grid resolution at which seed points can be projected to the nearest grid pixel without conflict. Following this, the algorithm marks the grid pixels which are destined to lie in the interior of the Voronoi regions and refines the boundaries of these regions. This algorithm uses an intrinsic quad tree-based approach. Like JFA, the proposed approach also requires $\log n$ steps to compute the Voronoi diagram for an $n \times n$ grid of pixels. But, for larger grids with fewer seed points (and hence large Voronoi regions), almost all the pixels will be marked as interior and hence will not be processed. This strategy enables both space optimization and better running times in practice. We explore the speed-ups obtained over JFA for different grid and seed set sizes. We implement the original JFA and FACET-JFA using CUDA to compute Voronoi diagrams in two and three dimensions. We report experimental results concerning the running time and other parameters across multiple GPU architectures.

As an application of FACET-JFA, we present a GPU accelerated technique for extraction of the channel network in biomolecules in two and three dimensions. The proposed method allows extraction of channels at real-time interactive rates and is thus suited for visual analysis of static and dynamic channel structures in Molecular Dynamics (MD) simulation trajectories. With examples, we demonstrate that the discrete representation and the use of FACET-JFA is appropriate for the typical resolutions required for visualizing the channels in MD trajectories at interactive rates.

6.1 Discrete Voronoi diagram computation

6.1.1 Definitions

Definition 6.1. *Let $[n]$ be the set $\{0, 1, \ldots, n-1\}$. A* grid *is defined as the Cartesian product $[n]^d$ of the set $[n]$. Here, d is the* dimension *of the grid and n is the* size *of the grid. A d-dimensional grid of size n is denoted by $[n]^d$. Any element $p \in [n]^d$ is a d-dimensional vector $(x_1, x_2, \ldots, x_d)$ and called a* pixel.

Definition 6.2. *Given a grid $[n]^d$ with a distance metric δ defined on it and a set of k seeds $S = \{s_1, s_2, \ldots, s_k\} \subseteq [n]^d$, the* discrete Voronoi diagram *is a function f defined as follows:*

$$f : [n]^d \to S$$
$$\textit{such that } f(p) = s_i \iff \forall j \neq i, \quad \delta(p, s_i) \leqslant \delta(p, s_j)$$

To make the above definition well defined, we will always assign lowest indexed seed to the pixel p whenever p is equidistant to multiple seeds. For any seed $s_i \in S$, $f^{-1}(s_i)$ is called the discrete Voronoi *region* of s_i.

Definition 6.3. *Let* $\Delta_1 = \{-1, 0, 1\}$. *The* neighbourhood *of a pixel* $p \in [n]^d$ *is the set of pixels* N_p, *defined as follows:*

$$N'_p = \{p + \delta_1 \quad \textit{such that} \quad \delta_1 \in \Delta_1^d\}$$
$$N_p = N'_p \cap [n]^d \setminus \{p\}$$

There are at most $3^d - 1$ pixels in the neighbourhood of a pixel $p \in [n]^d$. If the pixel lies on the boundary of the grid then size of neighbourhood sets is smaller than $3^d - 1$.

Definition 6.4. *A* cell *of size* l *at a pixel* $p \in [n]^d$ *is a smaller grid of size* l *at origin* p. *The set of pixels in the cell of size* l *at pixel* p, *denoted by* $C(p, l)$, *is given by:*

$$C(p, l) = \{p + \delta \ \textit{such that} \ \delta \in [l]^d\}$$

In 2D, a cell is a pixel in a grid of resolution $(n/l) \times (n/l)$. That is, every pixel in an $(n/l) \times (n/l)$ grid represents a cell in an $n \times n$ grid, each containing $l \times l$ pixels.

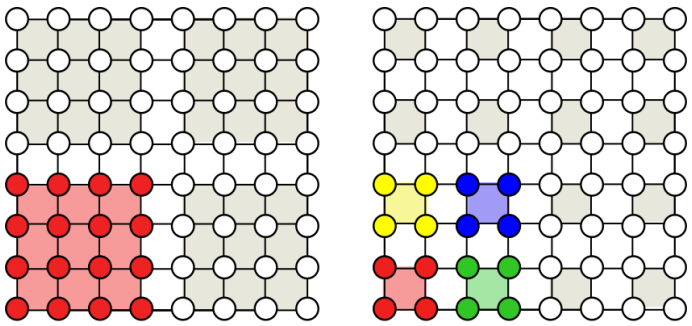


Figure 6.1: Left: A Cell $C(p, 4)$ in a grid $[8]^2$. Right: Its refinement into four cells of size 2.

Definition 6.5. *The* refinement *of a cell* $C(p, l)$, *denoted by* $R(C(p, l))$, *is the partition into* 2^d *equal sized cells. It is defined as follows:*

$$\Delta_{\frac{l}{2}} = \{0, \frac{l}{2}\}$$
$$R(C(p, l)) = \{C(p + \delta_{\frac{l}{2}}, \frac{l}{2}) \ \textit{such that} \ \delta_{\frac{l}{2}} \in \Delta_{\frac{l}{2}}^d\}$$

In 2D, refinement refers to increasing the resolution of the grid form an $(n/l) \times (n/l)$ grid to an $(2n/l) \times (2n/l)$ grid. Refer to Figure 6.1 for an example of a cell and its refinement.

Definition 6.6. *For a discrete Voronoi diagram f of a grid $[n]^d$, a pixel p is called a* boundary *pixel if there exists $q \in N_p$ such that $f(p) \neq f(q)$. The other pixels in the grid are called* interior *pixels.*

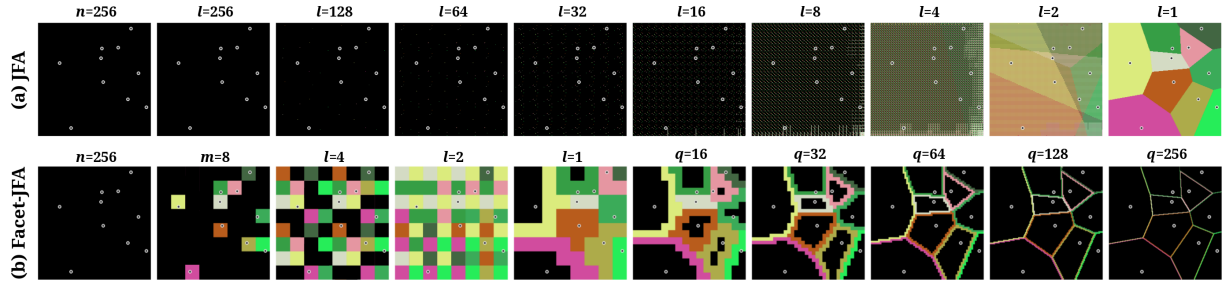


Figure 6.2: Comparison of execution of JFA and FACET-JFA on a 256×256 grid with 10 seed points. (a) JFA completes in 8 steps for this grid. In each step, all the pixels update their labels based on the seed closest to them. Notice that in earlier steps many pixels are coloured black as they have not received any valid seed till then. (b) FACET-JFA also takes 8 steps for completion. For this example, 8×8 grid is determined to be the initial resolution, on which JFA is executed resulting in coarse Voronoi diagram (frame with $l = 1$). In subsequent steps, the coarse boundaries are refined till the final resolution is reached. In refinement steps, the pixels coloured black are those which are marked and remain inactive, resulting in faster computation.

6.1.2 Jump Flooding and JFA

The idea of utilizing graphics hardware to compute geometric constructions is not new. The first attempt to compute Voronoi diagrams using graphics hardware was done by K. Hoff [52]. The algorithm described by Hoff uses projections of cones from seed points, drawing region boundaries where two cones meet. This was vastly improved in the more recent attempt by Rong *et al.* [84], using a parallel approach to flooding, known as Jump Flooding. This algorithm is based on the observation that while flooding an area with a label, each labelled pixel can transmit its label, instead of just the ones on the boundaries of the labelled region. This ensures an exponential growth in the number of labelled pixels in a grid and thus, can be computed in $O(\log n)$ time, for an $n \times n$ grid.

The work by Rong *et al.* [84] introduces two variants of the flooding algorithm, one with a halving step length and the other with a doubling step length. It is also shown that the former approach results in fewer errors, as compared to the latter one. The experiments in this chapter use the halving approach to flood label information. The jump flooding algorithm to compute Voronoi diagrams utilize this flooding. The JFA algorithm using jump flooding in the halving step mode proceeds in the following way:

1. Initially, each pixel corresponding to a seed s records a tuple $\langle s, position(s)\rangle$ and all other pixels record $\langle nil, nil\rangle$.

2. In step l, each pixel (x, y) passes the tuple corresponding to the seed closest to it to the pixels $(x + i, y + j), i, j \in \{-l, 0, l\}$. In cases when a pixel is closest to more than one seed, the tie is broken based on the indices of seeds.

3. The step size halves in each iteration and the above step is repeated starting with $l = n$ till $l = 1$ when the algorithm halts.

Figure 6.2(a) demonstrates how JFA proceeds on a 256×256 grid. There are several variants of JFA described in [84] and [85], including JFA+1, JFA+2, JFA^2 and 1+JFA. We use JFA+1, which is JFA with one additional round of flooding for the experiments in this chapter.

6.1.3 Facet-JFA

A great amount of work is done in the original JFA in the flooding of label information by the pixels which are in the interior of Voronoi regions. Much of this flooding can be seen as extraneous as it does not affect the boundaries of the Voronoi regions being formed. The proposed algorithm aims to eliminate this overhead by computing the region boundaries alone, instead of processing every pixel in each region. The proposed algorithm is presented as Algorithm 2 for 2D case. This algorithm can easily be extended to higher dimensions as well.

Just like JFA, the proposed algorithm also takes $\log n$ steps to finish. The algorithm marks the interior pixels, thus lowering the number of threads that need to be launched. This greatly reduces the total amount of work done by the algorithm and thus the running time, as demonstrated in the experiments. This algorithm can perform as bad as JFA if initial grid resolution m is close to or same as n. Figure 6.2 demonstrates the steps involved in Facet-JFA and how those steps compare with the steps involved in JFA. In the first $\log m$ steps, Facet-JFA essentially executes JFA to obtain the Voronoi diagram at a lower resolution of $m \times m$. In the next $\log n - \log m$ steps, the boundary cells are identified and refined iteratively. During refinement, a new pixel acquires a label corresponding to its parent or the label of the parent's neighbour. Executing single step of JFA with $l = 1$ accomplishes this refinement.

6.1.4 Runtime and space analysis

JFA and Facet-JFA both take $\log n$ steps to finish. Each of these steps involve processing multiple pixels in parallel. But the advantage of using Facet-JFA is that it processes smaller girds in initial stages, while at later stages when grid size becomes larger, many of the pixels are not processed. We perform qualitative assessment of Facet-JFA in terms of the number of pixels processed by the

Algorithm 2 FACET-JFA

Input: S: Set of seeds.
Input: n: Grid size.
Input: m: Initial grid resolution (optional).
Output: M: $n \times n$ grid where pixels on the Voronoi region boundary are set to the index of the seed.

1: If m is not provided, compute the smallest value $m = 2^p, p \in \mathbb{N}$ such that for every two seeds (i_1, j_1) and (i_2, j_2), $\frac{i_1 \cdot m}{n} \neq \frac{i_2 \cdot m}{n}$ or $\frac{j_1 \cdot m}{n} \neq \frac{j_2 \cdot m}{n}$.
2: M := Create an $m \times m$ grid.
3: Unmark each pixel in the $m \times m$ grid.
4: Run JFA on the $m \times m$ grid with seeds (i, j) replaced by $(\frac{i \cdot m}{n}, \frac{j \cdot m}{n})$.
5: Initialize q to m and repeat steps 6 to 9 doubling q after each iteration and halting when $q \geq n$.
6: M := Create a $2q \times 2q$ grid.
7: Refine each unmarked pixel (i, j) in the $q \times q$ grid to 4 pixels, $(2i, 2j), (2i + 1, 2j), (2i, 2j + 1), (2i + 1, 2j + 1)$ in the $2q \times 2q$ grid.
8: With $l = 1$, run one step of JFA on the grid M with seeds (i, j) replaced by $(\frac{i \cdot 2q}{n}, \frac{j \cdot 2q}{n})$. Launch threads only for the unmarked pixels.
9: Mark all pixels in M which have identically labelled neighbours. Again, launch threads only for the unmarked pixels.
10: **return** M

algorithm after performing JFA on the initial coarse grid. The lower the number of these pixels, the larger would be the performance gain, both in terms of timing benefit and potential space savings. We show that the algorithm processes $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1} \cdot \log n)$ pixels in the general case, and at most $5 \cdot (3k - 6) \cdot n \log n$ pixels in the specific case of a 2D grid. This is a factor of n improvement over JFA, which processes $\log n \cdot n^2$ pixels in its complete execution. Including the number of pixels processed in the JFA execution on the initial grid, the total number of pixels processed by FACET-JFA is at most $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1} \cdot \log n + m^2 \cdot \log m)$.

6.1.4.1 2D grid

Lemma 6.1. *The number of boundary pixels in the discrete Voronoi diagram of a two dimensional grid* $[n]^2$ *with seed set of size* k *is linear in* k *and* n*. The number of boundary pixels is at most* $5 \cdot (3k - 6) \cdot n$*.*

Proof. The Voronoi diagram is the dual of the Delaunay triangulation. So, number of edges in the 2D Voronoi diagram is equal to that in the Delaunay triangulation. The number of vertices in the Delaunay triangulation is exactly k i.e. the number of seeds. Since the Delaunay graph is planar, the number of edges is at most $(3k - 6)$.

Now, any line passing through $[n]^2$ can contribute at most $5n$ boundary pixels (Refer to Figure 6.3 for an example and Lemma 6.2 for proof in d dimensions). So, each Voronoi edge can

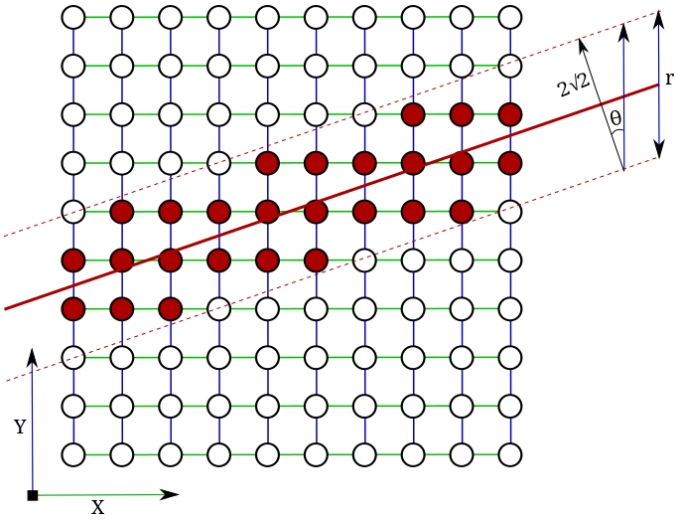


Figure 6.3: The set of boundary pixels (red) due to intersection of a line (red) with a $[10]^2$ grid.

contribute at most $5n$ boundary pixels. Therefore, the total number of boundary pixels in a two dimensional discrete Voronoi diagram is at most $5 \cdot (3k - 6) \cdot n$. □

Theorem 6.1. *Total number of pixels processed by* FACET-JFA *while computing the discrete Voronoi diagram for k seeds in a $[n]^2$ grid is bounded by $5 \cdot (3k - 6) \cdot n \log n$.*

Proof. FACET-JFA subdivides the grid in a quad tree fashion. The leaves of the quad tree are the pixels in the grid, while the internal nodes are the cells in the grid. The construction of the discrete Voronoi diagram of a grid using FACET-JFA involves reaching the boundary pixels of the Voronoi regions using this tree. By Lemma 6.1, we know that the number of boundary pixels in a discrete Voronoi diagram is at most $5 \cdot (3k - 6) \cdot n$. In a quad tree, to reach a leaf node we need to access $\log n$ internal nodes. So, during the course of a run of FACET-JFA, the total number of pixels processed is at most $5 \cdot (3k - 6) \cdot n \log n$. □

6.1.4.2 d-Dimensional grid

Lemma 6.2. *The number of boundary pixels resulting because of intersection of a hyperplane in $\mathbb{R}^d$ with the grid $[n]^d$ is at most $O(d \cdot n^{d-1})$.*

Proof. In $[n]^d$, all neighbours of a grid pixel are within a distance of $\sqrt{d}$. We consider two parallel hyperplanes at the distance of $\sqrt{d}$ from the given hyperplane. Refer to Figure 6.3 for an illustration. The given hyperplane is shown as bold red line while the two parallel hyperplanes are shown as dotted lines. Now, it is easy to see that boundary pixels can not lie outside the volume bounded by the two hyperplanes. So, estimating the maximum number of pixels that can lie in this volume will give an upper bound on the number of boundary pixels.

Given one of the d axes, the volume consists of at most n^{d-1} columns of pixels along this axis. We claim that for a suitable choice of axis the number of boundary pixels within each column is at most $(2d+1)$. Let r be the length of a column within the volume bounded by the two hyperplanes, see Figure 6.3. We choose the axis that minimizes the angle θ and maximizes $\cos\theta$. In this case, $\cos\theta \geq 1/\sqrt{d}$ where the equality holds when the hyperplane and its normal subtend the same angle with all the axes. Further, $r \cdot \cos\theta = 2\sqrt{d}$. So, the value of r is at most $2\sqrt{d} \times \sqrt{d} = 2d$. A column length of $2d$ within the volume corresponds to at most $(2d+1)$ pixels. Counting pixels within all columns results in an upper bound of $(2d+1) \cdot n^{d-1} = O(d \cdot n^{d-1})$ boundary pixels. □

For the case of $d = 2$, the number of boundary pixels due to a Voronoi edge is at most $5n$.

Lemma 6.3. *The number of boundary pixels in the discrete Voronoi diagram on a grid $[n]^d$ with seed set of size k is upper bounded by $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1})$.*

Proof. We know from Lemma 6.2 that intersection of any hyperplane can result in $O(d \cdot n^{d-1})$ boundary pixels in a grid. Also, it is known that there are $O(k^{\lceil \frac{d}{2} \rceil})$ Voronoi facets in a d-dimensional Voronoi diagram [27]. It follows that the total number of boundary pixels in a discrete Voronoi diagram is bounded by $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1})$. □

Theorem 6.2. *Total number of pixels processed by Facet-JFA for computation of discrete Voronoi diagram for k seeds in a $[n]^d$ grid is $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1} \cdot \log n)$.*

Proof. The argument given in Theorem 6.1 can be extended to the d dimensional case as well. By Lemma 6.3, we know that the number of boundary pixels in a discrete Voronoi diagram is $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1})$. In a d dimensional quad tree, to reach a leaf node we need to access $\log n$ internal nodes. Thus, the total number of pixels processed is at most $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1} \cdot \log n)$. □

6.1.5 Pre-processing in Facet-JFA: Computation of m

The computation of m in the first step of Facet-JFA is done on the CPU and is considered as a preprocessing step. In practice, m can be directly provided by the user, determined based on a user-defined error tolerance, or it can be domain specific and based on closest possible pair. Here we briefly discuss a few possible approaches to compute m.

Problem Given seed set $S = \{s_1, s_2, \cdots, s_k\}$ of size k and maximum grid size n. Compute the smallest grid size $m = 2^p$, which ensures projection of the k seeds without conflict. That is, no two seeds should share the same pixel in the $m \times m$ grid after projection.

We suggest following three solutions for this problem:

1. **Closest pair method:** The grid size that does not cause any conflicts may be computed by estimating the distance between the closest pair of seeds. The value of m can be computed in constant time given the closest pair distance.

2. **Domain specific:** The closest pair information can be domain specific. For example, in case of molecules, it is known that two atoms cannot be physically closer than a threshold distance. That threshold can be directly used to compute m in constant time. The domain expert can also provide an error tolerance threshold in terms of distance below which two seeds can be considered to be the same. This information can again be used to compute m in constant time.

3. **Brute force with grid:** We have to allocate an $n \times n$ grid for computation of Voronoi diagram using FACET-JFA. We can exploit this available space to determine m. For each seed point, set the pixel to which it is projected as 1. If after the projection it is found that the pixel is already set, then we have to consider higher values of m. Also, we can use binary search to identify a valid value of m. This approach can therefore determine m and simultaneously place the seeds in $O(k \log \log n)$ time.

The performance of above methods can be improved further by exploiting parallelism. For example, a brute force comparison of all pairs of seed points can be performed in parallel.

6.1.6 CUDA implementation

We implement jump flooding on CUDA using a gather-style approach, wherein in step l, each pixel gathers the label information from 8 neighbours l pixels away from it. It can easily be shown that this approach is equivalent to the halving mode of jump flooding. This approach is free from write-conflicts and the GPU based approach greatly benefits from this fact, as GPUs are inherently massively parallel. We used this gather-style approach to jump flooding to implement JFA and FACET-JFA.

In the implementation of JFA, the grid is copied only once to the device memory. A total of $\log n$ calls are made to the kernel, each call corresponding to a step of the flooding algorithm. The implementation of FACET-JFA involves multiple kernels as the algorithm consists of several phases. FACET-JFA requires the initial grid resolution m which is computed as described in previous section. Once m has been determined, FACET-JFA first computes Voronoi diagram at the coarsest resolution of $m \times m$ using JFA. Then the algorithm proceeds by repeatedly launching the cell refinement, JFA and marking steps in succession till the final resolution $n \times n$ is reached. At an intermediate grid resolution of $q \times q$, q^2 threads are launched, which return immediately after launching if the pixel is marked. Otherwise, they update the Voronoi region using one step ($l = 1$) of JFA on that grid. The cell refinements to obtain the $2q \times 2q$ grid and the marking of interior pixels is also done using CUDA kernels. The main overhead in this implementation lies in the launching of threads for every pixel, irrespective of whether it is marked or not. A better and thus more efficient

approach would be to launch threads only for the unmarked pixels, thus greatly reducing the launch overhead. Selective launching of threads only for unmarked pixels is non-trivial and not supported on all CUDA architectures.

It should be noted that space requirement in the current implementation is n^2 which is used for storing the $n \times n$ matrix. The refinement step which results in doubling of resolution is handled carefully within the allocated space so that no extra space is required even temporarily. Theoretically, FACET-JFA requires much less space than JFA as discussed in 6.1.4, but achieving that limit would require special indexing structures.

6.2 Experimental results

6.2.1 Experimental setup

The experiments have been performed simultaneously on two different GPU architectures from nVidia, namely the latest Kepler and the older Fermi. The GPUs used for the experiments are the GTX-660 Ti and Tesla C2050. The CPU in both cases was an Intel Xeon octa core, running at 2GHz and with 16 GB of main memory. The following experiments were done.

1. Time comparisons for JFA and FACET-JFA in a 2D case on the Kepler and Fermi architectures (Tables 6.1 and 6.2).

2. Time comparisons for JFA and FACET-JFA in a 3D case on the Kepler architecture (Table 6.3).

3. Comparison of the number of threads launched in JFA and FACET-JFA (Table 6.4).

The experiments were run at resolutions ranging from 256×256 to 4096×4096, with the resolution doubling in each run. The number of seeds was increased from 10 to 100,000 incrementally for each resolution. The default random number generator, `rand()` was used to generate the seed locations. Running times are reported in milliseconds and are calculated separately for the actual kernel execution and copying of the grid from the host to the device and back. In the case of FACET-JFA, the time taken to generate the starting resolution is taken as a preprocessing step and is not included in the timing computation. In each experiment, the time reported is the average taken over 100 runs of the algorithm.

6.2.2 Observations

The speed-up obtained on the GTX-660Ti GPU has been plotted in Figure 6.4. The performance of FACET-JFA was superior to that of JFA in most cases, with speed-ups as high as 10x in the case of 10 seeds on a 4096×4096 grid on the GTX-660Ti. It was observed that the number of seeds and the

Table 6.1: Timing results for FACET-JFA and JFA for 2D grids on nVidia GTX-660 Ti

Grid size	#Seeds	FACET-JFA time (ms)			JFA time (ms)			Total	Exec.
(n)	(k)	Mem.	Exec.	Total	Mem.	Exec.	Total	Speed-up	Speed-up
256	10	1.20	0.34	1.54	1.22	0.55	1.77	1.15	1.61
	100	1.21	0.42	1.62	1.20	0.58	1.78	1.10	1.40
	500	1.23	1.11	2.34	1.25	1.10	2.35	1.01	0.99
	1000	1.28	1.31	2.58	1.24	1.39	2.62	1.02	1.06
512	10	1.81	0.66	2.47	1.79	2.08	3.88	1.57	3.17
	100	1.81	1.21	3.02	1.77	2.24	4.01	1.33	1.86
	500	1.82	3.87	5.69	1.75	3.41	5.16	0.91	0.88
	1000	1.77	4.41	6.19	1.80	4.19	6.00	0.97	0.95
1024	10	3.62	1.34	4.96	3.58	8.04	11.61	2.34	5.99
	100	3.61	2.43	6.04	3.51	8.39	11.90	1.97	3.45
	500	3.63	7.83	11.46	3.49	12.05	15.54	1.36	1.54
	1000	3.67	14.64	18.31	3.49	14.37	17.86	0.98	0.98
	10000	3.63	23.77	27.40	3.48	22.99	26.46	0.97	0.97
2048	10	10.66	3.73	14.38	9.75	28.95	38.70	2.69	7.77
	100	10.46	6.59	17.05	9.60	31.37	40.97	2.40	4.76
	500	10.38	15.33	25.71	9.74	47.83	57.57	2.24	3.12
	1000	10.27	27.62	37.89	9.74	57.02	66.76	1.76	2.06
	10000	10.38	90.38	100.76	10.71	95.38	106.09	1.05	1.06
4096	10	36.10	12.13	48.22	35.48	131.88	167.36	3.47	10.87
	100	35.77	18.52	54.28	35.38	140.51	175.89	3.24	7.59
	500	35.73	37.70	73.43	35.54	199.05	234.58	3.19	5.28
	1000	36.07	67.49	103.56	35.22	236.12	271.34	2.62	3.50
	10000	35.52	368.32	403.84	35.62	331.23	366.85	0.91	0.90
	100000	36.90	506.09	542.98	35.62	494.46	530.08	0.98	0.98

speed-up observed are inversely related, as the presence of a larger number of seeds would generally result in smaller regions, resulting in fewer interior points. This could also be due to the fact that, in FACET-JFA, the starting resolution of a grid would increase in proportion to the number of seeds, thus necessitating larger numbers of threads to be launched in the initial call to JFA. Nevertheless, in cases where JFA outperformed Facet JFA, the slow down was marginal (0.9x). The threshold on the number of seeds for which JFA starts to outperform FACET-JFA was experimentally observed to be approximately n seeds for an $n \times n$ grid.

The pattern of speed-up observed on the Tesla C2050 almost exactly mirrored the observations

Table 6.2: Timing results for FACET-JFA and JFA for 2D grids on nVidia Tesla C2050

Grid size	#Seeds	FACET-JFA time (ms)			JFA time (ms)			Total	Exec.
(n)	(k)	Mem.	Exec.	Total	Mem.	Exec.	Total	Speed-up	Speed-up
256	10	3.08	0.40	3.47	2.97	0.46	3.44	0.99	1.17
	100	2.96	0.45	3.41	2.89	0.54	3.43	1.00	1.20
	500	2.84	0.75	3.60	3.06	0.66	3.72	1.03	0.88
	1000	2.96	0.84	3.79	2.98	0.73	3.72	0.98	0.88
512	10	3.64	0.77	4.41	3.90	1.76	5.66	1.28	2.28
	100	3.45	1.01	4.47	3.54	2.12	5.66	1.27	2.09
	500	4.00	2.89	6.89	3.65	2.47	6.12	0.89	0.85
	1000	3.44	3.22	6.66	3.90	2.73	6.62	0.99	0.85
1024	10	4.98	1.75	6.73	4.90	7.37	12.27	1.82	4.22
	100	5.75	2.67	8.42	5.11	8.69	13.80	1.64	3.25
	500	5.97	11.10	17.07	5.54	10.36	15.90	0.93	0.93
	1000	4.98	13.02	18.00	5.18	11.77	16.95	0.94	0.90
	10000	5.08	21.54	26.62	5.15	23.27	28.42	1.07	1.08
2048	10	13.67	4.90	18.56	13.40	30.72	44.12	2.38	6.28
	100	13.92	8.24	22.16	13.27	35.87	49.14	2.22	4.35
	500	14.45	13.87	28.32	13.93	40.78	54.71	1.93	2.94
	1000	13.42	28.19	41.62	14.94	46.50	61.44	1.48	1.65
	10000	13.67	89.31	102.98	13.28	84.74	98.01	0.95	0.95
4096	10	45.29	16.18	61.47	43.53	136.62	180.15	2.93	8.44
	100	43.53	22.20	65.73	43.77	153.36	197.12	3.00	6.91
	500	45.51	35.92	81.43	43.54	177.32	220.85	2.71	4.94
	1000	41.69	68.41	110.10	53.50	199.25	252.75	2.30	2.91
	10000	53.90	361.26	415.16	43.49	372.48	415.97	1.00	1.03
	100000	45.53	573.48	619.01	44.16	607.98	652.13	1.05	1.06

on the GTX-660Ti, albeit at a slightly lower factor. This can be attributed to fewer number of cores on the Tesla (448, as compared to 1344 on the GTX-660Ti). We also observe similar speed-ups in the 3-dimensional case. For example, for a 512^3 grid with 1000 seeds, we observed a speed-up of around 6x. The time taken to copy the contents of the host memory to the device and back dominated the total running time in the 256×256 and 512×512 grids, whereas the time taken to execute the kernel assumed prominence from the 1024×1024 grid onwards. The speed-up was most pronounced, as expected, for the largest grid.

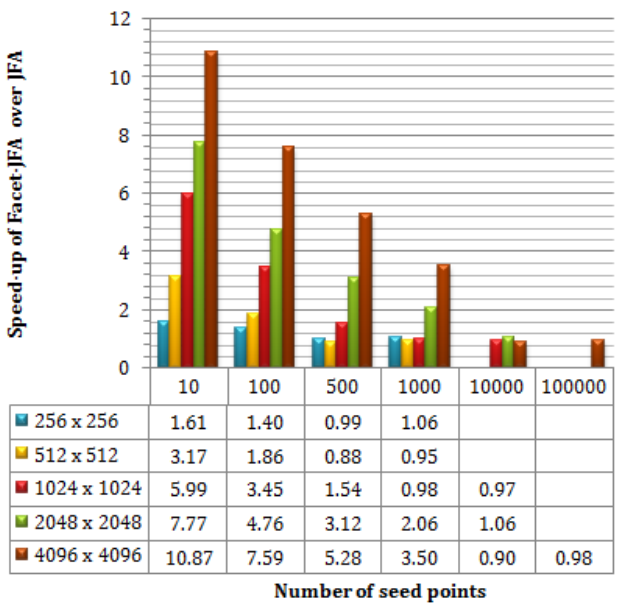
Refer to Table 6.4. We note a marked improvement in the number of threads launched (fewer

Table 6.3: Timing results for FACET-JFA and JFA for 3D grids on nVidia GTX-660 Ti

Grid size	#Seeds	FACET-JFA time (ms)			JFA time (ms)			Total	Exec.
(n)	(k)	Mem.	Exec.	Total	Mem.	Exec.	Total	Speed-up	Speed-up
32	10	1.20	0.63	1.83	1.46	0.90	2.35	1.29	1.44
	100	1.19	0.71	1.90	1.49	1.14	2.64	1.39	1.62
	500	1.19	1.59	2.78	1.16	1.88	3.04	1.09	1.18
	1000	1.22	2.18	3.39	1.30	2.18	3.48	1.03	1.00
64	10	2.13	2.27	4.40	1.91	7.38	9.28	2.11	3.26
	100	1.91	2.86	4.76	1.93	6.96	8.89	1.87	2.44
	500	1.92	10.18	12.10	1.94	10.42	12.36	1.02	1.02
	1000	1.88	11.66	13.54	2.15	12.08	14.23	1.05	1.04
	100000	2.01	18.95	20.96	1.98	18.80	20.77	0.99	0.99
128	10	6.33	9.45	15.78	5.76	52.20	57.96	3.67	5.52
	100	6.01	16.73	22.74	6.01	52.30	58.31	2.56	3.13
	500	5.88	37.08	42.96	5.77	75.28	81.05	1.89	2.03
	1000	5.98	90.34	96.32	5.71	86.63	92.34	0.96	0.96
	10000	6.12	131.57	137.69	5.97	128.20	134.18	0.97	0.97
	100000	6.29	197.60	203.89	6.41	194.46	200.87	0.99	0.98
256	10	38.60	41.48	80.09	39.53	422.78	462.31	5.77	10.19
	100	38.89	73.31	112.20	39.30	427.80	467.09	4.16	5.84
	500	40.37	138.50	178.87	39.20	603.65	642.85	3.59	4.36
	1000	38.65	263.23	301.89	38.68	681.13	719.81	2.38	2.59
	10000	37.59	1007.48	1045.07	39.72	987.67	1027.39	0.98	0.98
	100000	35.13	1394.19	1429.32	38.18	1372.33	1410.51	0.99	0.98
512	10	278.03	179.64	457.67	277.91	3616.64	3894.55	8.51	20.13
	100	283.79	305.84	589.63	250.35	3694.71	3945.06	6.69	12.08
	500	283.55	514.93	798.47	263.49	5114.14	5377.63	6.73	9.93
	1000	248.13	974.90	1223.02	259.58	5700.99	5960.56	4.87	5.85
	10000	245.70	8137.42	8383.12	260.08	7952.83	8212.91	0.98	0.98
	100000	255.63	10812.28	11067.91	254.95	10669.75	10924.69	0.99	0.99

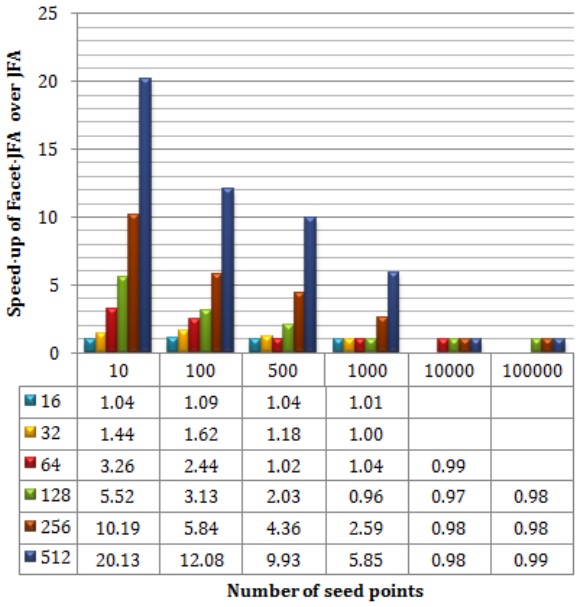
than JFA) in most of the cases under consideration. The trend we observed in this experiment almost exactly mirrors the trends observed in the speed-up, implying that speed-up is a direct consequence of the number of threads launched. However, we observed that the speed-up is a scaled down value of the ratio of the improvements in the number of launched threads (*e.g.* 88 times fewer active threads translated into 11x speed-up).

Comparison of three parallel algorithms. In Figure 6.5 we compare, in detail, the FACET-JFA runtime performance against JFA and the parallel brute-force algorithm. In the brute-force algorithm, each pixel iterates over the seed set and finds the seed closest to itself. All pixels are processed



	10	100	500	1000	10000	100000
256 x 256	1.61	1.40	0.99	1.06		
512 x 512	3.17	1.86	0.88	0.95		
1024 x 1024	5.99	3.45	1.54	0.98	0.97	
2048 x 2048	7.77	4.76	3.12	2.06	1.06	
4096 x 4096	10.87	7.59	5.28	3.50	0.90	0.98

(a) 2D case



	10	100	500	1000	10000	100000
16	1.04	1.09	1.04	1.01		
32	1.44	1.62	1.18	1.00		
64	3.26	2.44	1.02	1.04	0.99	
128	5.52	3.13	2.03	0.96	0.97	0.98
256	10.19	5.84	4.36	2.59	0.98	0.98
512	20.13	12.08	9.93	5.85	0.98	0.99

(b) 3D case

Figure 6.4: Facet-JFA speed-ups in 2D and 3D case.

Figure 6.5: Comparison of Facet-JFA with JFA and brute-force algorithm.

Table 6.4: Unmarked pixels in FACET-JFA for 2D case and the *savings ratio*

Grid size (n)	#Seeds (k)	Unmarked pixels	Total pixels	Savings ratio
256	10	10,260	65,536	6.39
	100	36,300	65,536	1.81
	500	65,536	65,536	1.00
	1,000	65,536	65,536	1.00
512	10	19,801	262,144	13.24
	100	83,055	262,144	3.16
	500	258,289	262,144	1.01
	1,000	262,144	262,144	1.00
1024	10	50,180	1,048,576	20.90
	100	180,754	1,048,576	5.80
	500	552,137	1,048,576	1.90
	1,000	1,048,576	1,048,576	1.00
	10,000	1,048,576	1,048,576	1.00
2048	10	92,772	4,194,304	45.21
	100	376,677	4,194,304	11.14
	500	909,660	4,194,304	4.61
	1,000	2,322,336	4,194,304	1.81
	10,000	4,194,304	4,194,304	1.00
4096	10	190,093	16,777,216	88.26
	100	796,206	16,777,216	21.07
	500	2,420,779	16,777,216	6.93
	1,000	5,595,679	16,777,216	3.00
	10,000	16,777,216	16,777,216	1.00
	100,000	16,777,216	16,777,216	1.00

in parallel. The running time of this algorithm is linear in k, the seed set size. In this experiment, we fixed the grid size n to be 1024, while k was varied from 32 to 2048 with a step size of 32. The execution times for the three algorithms were averaged over 100 runs for each value of k. The seeds were placed randomly in the grid for each run, and all the algorithms processed the same input data.

From the plot in Figure 6.5, it can be observed that for very small values of $k < 150$, the brute-force algorithm performs better (the blue region). For higher values of $k > 1024$, FACET-JFA performance is similar to or slightly worse than that of JFA (the red region). The window in which FACET-JFA out-performs the other two algorithms $(150 < k < 1024)$ is coloured green. Designing an automatic decider to select an algorithm based on the values of k and n is an interesting challenge that can be explored in future.

6.3 Application to biomolecular channel extraction

A *channel* is a pathway through the empty space within a molecule that connects an internal point and the molecular exterior [80]. Channels are crucial for the migration of ions, solvent, and small molecules through proteins, and their ultimate binding to the functional sites. Channels through transmembrane proteins selectively transport ions and small molecules across cell membrane. In recent years, continuous Voronoi diagrams have been used to determine center-lines of the channels in biomolecules [68, 81, 100, 75]. The set of all channel center-lines is usually referred to as the *channel network* of the biomolecule.

Biomolecules are not static entities, they undergo various structural changes dynamically which are important for their function. Molecular Dynamics (MD) simulation trajectories are series of snapshots of the biomolecule as it undergoes changes over time. This simulation data has proved crucial in understanding dynamic behaviour of biomolecules. Recently, there has been great interest in development of fast techniques for analysis of MD trajectories [59, 78]. Lindow *et al.* [69] addressed the problem of channel extraction in MD trajectories using continuous Voronoi diagrams with focus more on the accuracy of detected channels rather than interactive analysis. Discrete Voronoi diagrams can be utilized to overcome the time-consuming step of computing continuous Voronoi diagram, and the extracted discrete channel network is sufficient for visual analysis. Further, here we are interested in computation of Voronoi edges only, which makes FACET-JFA an ideal candidate.

In this section, we present a GPU accelerated technique for computation of channel center-lines in biomolecules. We motivate this problem and present our technique using 2D synthetic data. However, the method can be easily extended to 3D. We show results both for 2D and 3D data. We exploit the fast computation of Voronoi facets by FACET-JFA for extracting channel network in the biomolecule. The Voronoi edges provide the locus of points which are locally farthest from the closest pair of atoms. The Voronoi edges can be restricted to empty regions inside the molecule to obtain the channel network. The discrete Voronoi diagram is computed using the GPU accelerated FACET-JFA. We additionally propose parallel methods for all other stages of the channel extraction algorithm. This implementation can process molecules of considerable size at a grid size suited for fast volume rendering on modern GPUs. Thus, using the proposed method, it is possible to interactively analyse static as well as dynamic channel structures in MD trajectories.

6.3.1 Channel extraction algorithm

Algorithm 3 describes the proposed parallel approach for fast extraction of channel network. To simplify notation, we refer to disks and union of disks in the 2D data also as atoms and molecules,

respectively. The working of this algorithm is demonstrated with a 2D example in Figure 6.6. Brief demonstration for a 3D example is provided in Figure 6.7.

Algorithm 3 Extract Channel Network

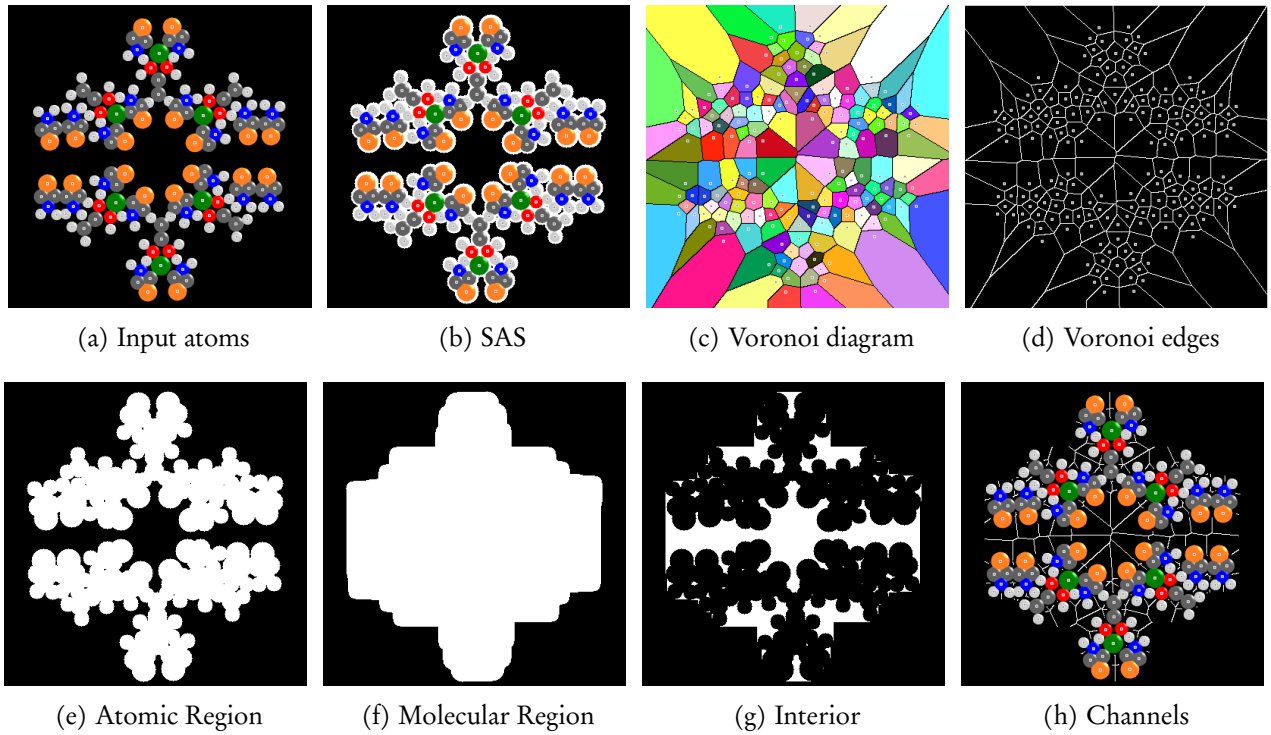
Input: S: Set of atoms.
Input: r_s: Solvent radius.
Input: n: Grid size.
Output: CN: $n \times n$ grid where pixels on channel center-lines are set to 1 while other pixels are 0.
Output: AM: $n \times n$ grid where pixels occupied by atoms of biomolecule are set to 1. This is an optional output.

1: VD := Construct discrete Voronoi diagram for S using FACET-JFA.
2: VE := Extract Voronoi edges by processing each pixel in VD in parallel. Set such edge pixels to 1. This step is not required in 2D, as VD already consists of only edges.
3: AM := Process each atom $a \in$ S in parallel and set pixels lying inside the ball of radius $r_a + r_s$ to 1. AM is called *atomic region mask*.
4: Shoot n rays in X direction in parallel into AM and determine their first and last intersections with AM. Repeat the same procedure for Y direction.
5: Construct MM, the *molecular region mask* where the pixel is set to 1 if they lie inside at least one of the intersection intervals determined in previous step.
6: IM := AM XOR MM. IM is called *molecular inside mask*.
7: CN := VE AND IM. Restrict Voronoi edges to inside region of molecule to obtain the *channel network* for the molecule.
8: **return** CN, AM

6.3.2 Discussion

As mentioned earlier, the proposed GPU accelerated extraction of channels is particularly suited for gaining a quick overview of channels in Molecular Dynamics (MD) trajectories. With fast channel extraction, the user can view the evolution of channels over time in MD data which typically consists of thousands of steps, and identify critical time steps. Further, channels can be computed for different solvent (probe) radii at interactive rates. An example use of this technique for study of dynamic channels in a 2D synthetic MD trajectory is shown in Figure 6.9.

Figure 6.8 shows the quality of results obtained for biomolecules with number of atoms ranging from few hundreds to thirteen thousand. The grid resolution used is $128 \times 128 \times 128$ which is enough for computing a good overview of the channel network in the molecule. This resolution is ideal for high quality volume rendering at interactive rates on modern graphics hardware. The results shown in Figure 6.8 can be computed and simultaneously visualized at interactive speed of 10 FPS. Currently we don't support analysis of the extracted channels, like identification of connected components or pruning of small channels. This facility can be introduced to give the user a richer



(a) Input atoms (b) SAS (c) Voronoi diagram (d) Voronoi edges

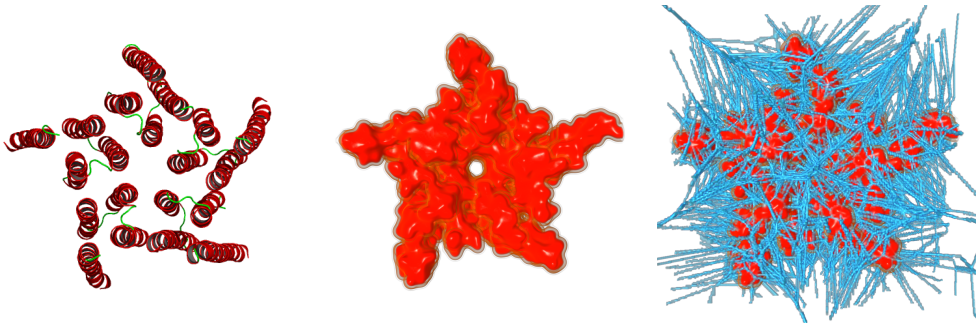
(e) Atomic Region (f) Molecular Region (g) Interior (h) Channels

Figure 6.6: Demonstration of GPU accelerated extraction of biomolecular channel network. (a) A synthetic molecule in 2D. (b) Solvent accessible surface constructed by incrementing the atomic radii by the solvent radius. (c) Voronoi diagram for set of atom centres. Each Voronoi region is given a random colour. (d) Voronoi edges. (e) The *atomic region mask* obtained by projecting each atom on the grid. This mask is computed by processing the atoms in parallel. (f) The *molecular region mask* obtained by shooting rays on atomic region mask. We shoot n rays from the bottom and the left, and determine each ray's first and last intersection points with the atomic region mask. Again parallelism is exploited by processing rays in parallel. (g) Pixels in the *molecular interior* are determined by performing XOR operation on atomic region mask and molecular region mask. The pixels in molecular interior are exactly those which lie inside the molecular region but are not occupied by atoms of the molecule. (h) Finally, the Voronoi edges are restricted to the region inside the molecule to obtain the *channel network*. This is accomplished by simply performing AND operation on Voronoi edge mask and molecular interior mask.

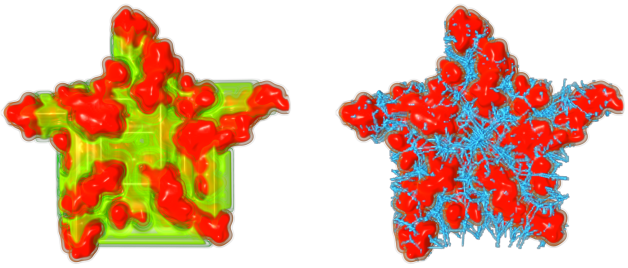
experience. We believe this minimal analysis can also be performed at interactive rates.

6.4 Conclusions

We proposed a variant of JFA, Facet-JFA, to compute discrete Voronoi diagrams using a GPU, which optimises on the number of threads launched, and hence the running time. The proposed algorithm has been studied both theoretically and via experiments on large data sizes. Several improvements are possible over the proposition. Facet-JFA uses a quad tree to refine the starting grid.

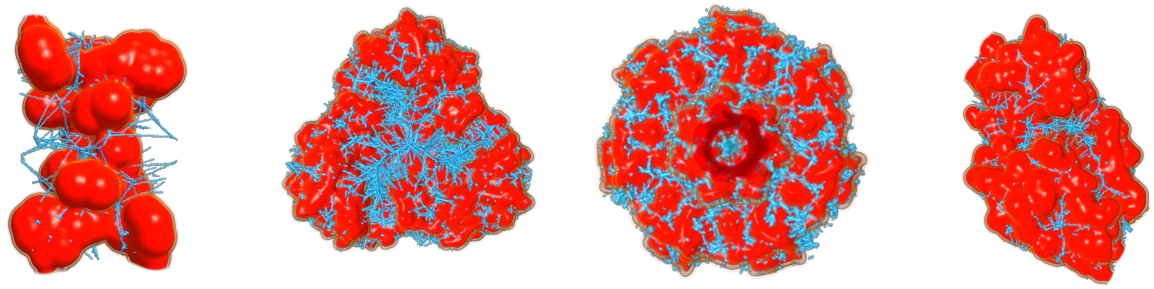


(a) 1OED (b) Atomic region (c) Voronoi edges



(d) Interior (e) Channels

Figure 6.7: Extension of 2D channel extraction algorithm to 3D. (a) A trans-membrane channel protein, PDB id: 1OED, is shown in cartoon representation. (b) The atoms are projected onto the 3D gid to determine atomic region. (c) The Voronoi edges are computed using FACET-JFA. (d) The region inside the molecule (shown in yellow) is determined by shooting rays in 3 orthogonal directions. (e) The Voronoi edges are restricted to the molecular inside region resulting in highlighting the molecular channel network.



(a) 1GRM (184) (b) 2J1N (8,142) (c) 2OAU (13,573) (d) 200L (1,289)

Figure 6.8: A few channel networks extracted are shown in blue along with the atomic region (orange). These images were generated using 128^3 grid resolution at interactive speeds. Number of atoms are in the brackets.

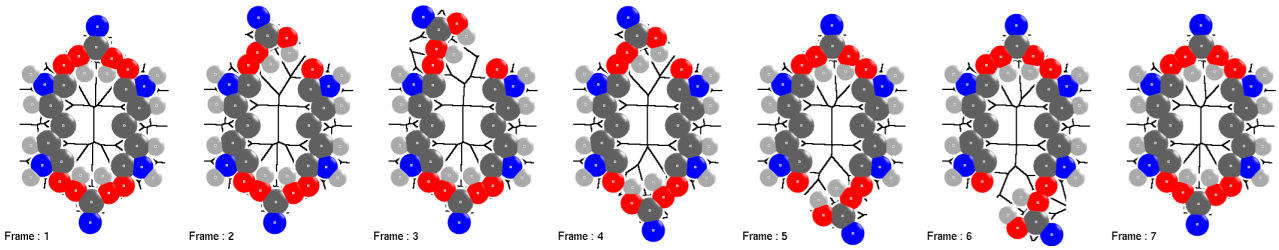


Figure 6.9: An example of dynamic channel in Molecular Dynamics simulation trajectory. The molecule in this synthetic dataset has two gates which are both closed initially. In Frame 2, the gate at the top opens resulting in channel to central cavity. In Frame 5, the top gate closes completely while bottom gate starts to open revealing a dynamic channel from top to the bottom. By Frame 7, the molecule regains its closed state.

The starting grid could have been non-uniform, thus starting with varying levels of refinement in various regions. This would require a divide and conquer approach to merge regions with the same quad-tree level, resulting in the launch of fewer threads, as compared to Facet-JFA. Also, in most real world applications, the computation of approximate Voronoi boundaries might suffice. Hence, fewer iterations of the refinement steps of Facet-JFA might provide an acceptable resolution for the application.

Chapter 7

Parallel Computation of Alpha Complex for Biomolecules

Alpha complex, a subset of the Delaunay triangulation, has been extensively used as the underlying representation for biomolecular structures. We propose a GPU based parallel algorithm for the computation of the alpha complex, which exploits the knowledge of typical spatial distribution and sizes of atoms in a biomolecule. Unlike existing methods, this algorithm does not require prior construction of the Delaunay triangulation. The algorithm computes the alpha complex in two stages. The first stage proceeds in a bottom up fashion and computes a superset of the edges, triangles, and tetrahedra belonging to the alpha complex. The false positives from this estimation stage are removed in a subsequent pruning stage to obtain the correct alpha complex. Computational experiments on several biomolecules demonstrate the superior performance of the algorithm, upto $50\times$ when compared to existing methods that are optimized for biomolecules.

7.1 Introduction

The alpha complex of a set of points in $\mathbb{R}^3$ is a subset of the Delaunay triangulation. A size parameter α determines the set of simplices (tetrahedra, triangles, edges, and vertices) of the Delaunay triangulation that are included in the alpha complex. It is an elegant representation of the shape of the set of points [36, 35, 39] and has found various applications, particularly in molecular modeling and molecular graphics. The atoms in a biomolecule are represented by weighted points in $\mathbb{R}^3$ and the region occupied by the molecule is represented by the union of balls centered at these points. The geometric shape of a biomolecule determines its function, namely how it interacts with other biomolecules. The alpha complex represents the geometric shape of the molecule very efficiently. It has been widely used for computing and studying geometric features such as cavities and chan-

nels [65, 66, 30, 75, 90, 73, 60]. Further, an alpha complex based representation is also crucial for accurate computation of geometric properties like volume and surface area [64, 34, 72].

Advances in imaging technology has resulted in a significant increase in the size of molecular structure data. This necessitates the developments of efficient methods for storing, processing, and querying these structures. In this chapter, we study the problem of efficient construction of the alpha complex with particular focus on point distributions that are typical of biomolecules. In particular, we present a parallel algorithm for computing the alpha complex and an efficient GPU implementation that outperforms existing methods. In contrast to existing algorithms, our algorithm does not require the explicit construction of the Delaunay triangulation.

7.1.1 Related work

The Delaunay triangulation has been studied within the field of computational geometry for several decades and numerous algorithms have been proposed for its construction [4]. Below, we describe only a few methods that are most relevant. A tetrahedron belongs to the Delaunay triangulation of a set of points in $\mathbb{R}^3$ if and only if it satisfies the *empty circumsphere* property, namely no point is contained within the circumsphere of the tetrahedron. The Bowyer-Watson algorithm [11, 99] and the incremental insertion algorithm by Guibas and Stolfi [47] are based on the above characterization of the Delaunay triangulation. In both methods, points are inserted incrementally and the triangulation is locally updated to ensure that the Delaunay property is satisfied. The incremental insertion method followed by bi-stellar flipping works in higher dimensions also [43] and can construct the Delaunay triangulation in $O(n \log n + n^{\lceil d/2 \rceil})$ time in the worst case, where n is the number of input points in $\mathbb{R}^d$. A second approach to constructing the Delaunay triangulation is based on its equivalence to the convex hull of the points lifted onto a $(d+1)$-dimensional paraboloid [42].

A third divide-and-conquer approach partitions the inputs points into two or multiple subsets, constructs the Delaunay triangulation for each partition, and merges the pieces of the triangulation finally. The merge procedure depends on the ability to order the edges incident on a vertex and hence works only in $\mathbb{R}^2$. Extension to $\mathbb{R}^3$ requires that the merge procedure be executed first [22]. The divide-and-conquer strategy directly extends to a parallel algorithm [77, 15]. The DeWall algorithm [22] partitions the input point set into two halves and first constructs the triangulation of points lying within the boundary region of the two partitions. The Delaunay triangulation of the two holes is then constructed in parallel. The process is repeated recursively resulting in increased parallelism. Cao et al. [15] have developed a GPU parallel algorithm, *gDel3D*, that constructs the Delaunay triangulation in two stages. In the first stage, points are inserted in parallel followed by flipping to obtain an approximate Delaunay triangulation. In the second stage, a star splaying pro-

cedure works locally to convert non-Delaunay tetrahedra into Delaunay tetrahedra. The algorithm can be extended to construct the weighted Delaunay triangulation for points with weights. Cao et al. report upto $10\times$ speed up over a sequential implementation for constructing the weighted Delaunay triangulation of 3 million weighted points.

All existing algorithms for constructing the alpha complex [35, 72, 25] require that the Delaunay triangulation be computed first. Simplices that belong to the alpha complex are identified using a size filtration in a second step. In the case of biomolecules, the size of the alpha complex is a small fraction of the size of the Delauanay triangulation and hence the Delaunay triangulation construction is often the bottleneck in the alpha complex computation. The key difficulty lies in the absence of a direct characterization of simplices that belong to the alpha complex.

7.1.2 Summary of results

We propose an algorithm that avoids the expensive Delaunay triangulation computation and instead directly computes the alpha complex for biomolecules. Key contributions of this chapter are summarized below:

- A new characterization of the alpha complex – a set of conditions necessary and sufficient for a simplex to be a part of the alpha complex.
- A new algorithm for computing the alpha complex of a set of weighted points in $\mathbb{R}^3$. The algorithm identifies simplices of the alpha complex in decreasing order of dimension without computing the complete weighted Delaunay triangulation.
- An efficient CUDA based parallel implementation of this algorithm for biomolecular data that can compute the alpha complex for a 10 million point dataset in approximately 10 seconds.
- A proof of correctness of the algorithm and comprehensive experimental validation to demonstrate that it outperforms existing methods.

While the experimental results presented here focus on biomolecular data, the algorithm is applicable to data from other application domains as well. In particular, the efficient GPU implementation may be used for points that arise in smoothed particle hydrodynamics (SPH) simulations, atomistic simulations in material science, and particle systems that appear in computational fluid dynamics (CFD).

7.2 Background

Although we discussed the mathematical definitions in detail previously in Chapter 2, here we provide a brief summary again with additional details relevant for this chapter.

We review the necessary background on Delaunay triangulations required to describe the algorithm and also establish a new characterization of the alpha complex that does not require the Delaunay triangulation. For a detailed description of Delaunay triangulations, alpha complexes, and related structures, we refer the reader to various books on the topic [4, 38, 32].

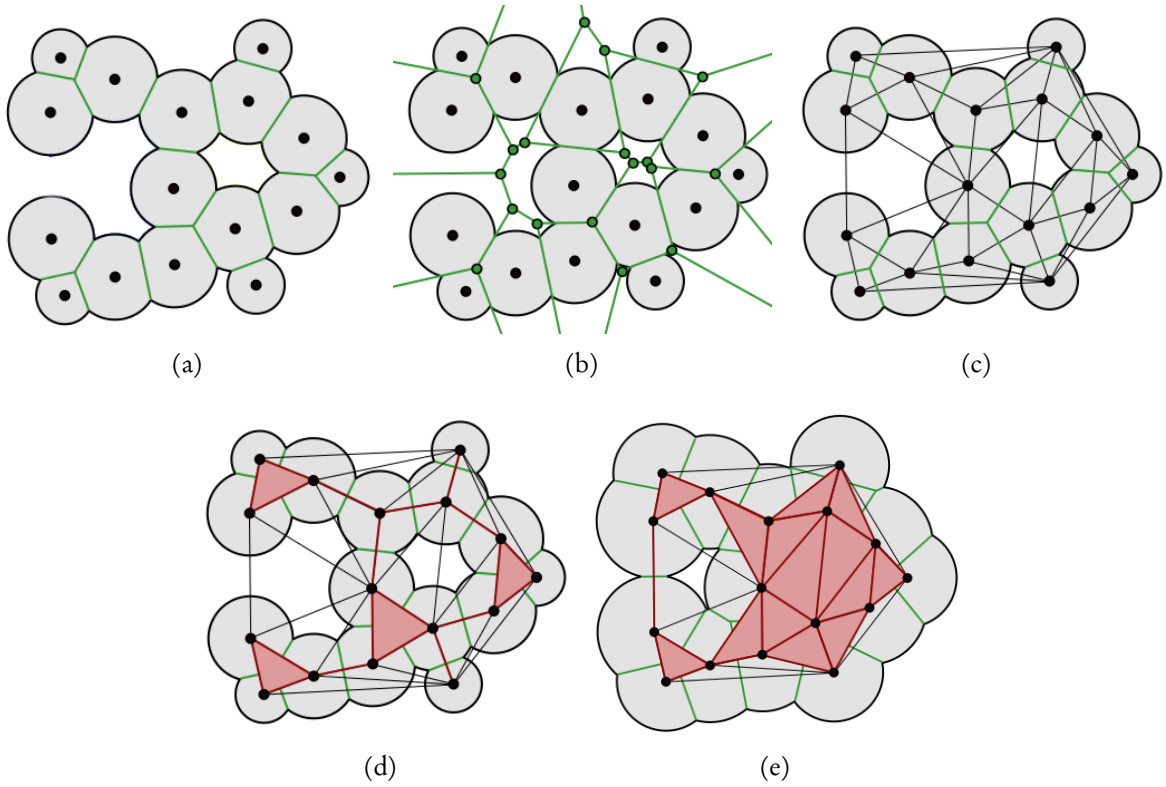


Figure 7.1: 2D weighted Delaunay triangulation and alpha complex. (a) A set of weighted points B in $\mathbb{R}^2$ shown as disks. (b) The weighted Voronoi diagram of B. Voronoi edges and vertices are highlighted in green. (c) The weighted Delaunay complex is the dual of the weighted Voronoi diagram. (d) The alpha complex K_α for $\alpha = 0$ is shown in red. This is the dual of the intersection of the weighted Voronoi diagram and union of balls. (e) The alpha complex shown for an $\alpha > 0$. It is the dual of the intersection of the weighted Voronoi diagram and union of balls after growing them to have radius $\sqrt{r_i^2 + \alpha}$.

Let $B = \{b_i\}$ denote a set of balls or weighted points, where $b_i = (p_i, r_i)$ represents a ball centered at p_i with radius r_i. We limit our discussion to balls in $\mathbb{R}^3$, so $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$. Further, we assume that the points in B are in general position, *i.e.*, no 2 points have the same location, no 3

points are collinear, no 4 points are coplanar, and no subset of 5 points are equidistant from a point in $\mathbb{R}^3$. Such configurations are called degeneracies. In practice, a degenerate input can be handled via symbolic perturbation [41].

7.2.1 Simplex and simplicial complex

A d-dimensional *simplex* σ^d is defined as the convex hull of $d+1$ affinely independent points. Assuming the centres of balls in B are in general position, all $(d+1)$ sized subsets of B form a simplex $\sigma^d = (p_0^\sigma, p_1^\sigma, \ldots, p_d^\sigma)$. For simplicity, we sometimes use b_i instead of the center p_i to refer to points incident on a simplex. For example, we may write $\sigma^d = (b_0^\sigma, b_1^\sigma, \ldots, b_d^\sigma)$.

A non-empty strict subset of σ^d is also a simplex but with dimension smaller than d. Such a simplex is called a *face* of σ^d. Specifically, a $(d-1)$-dimensional face of σ^d is referred to as a *facet* of σ^d. A set of simplices K is called a *simplicial complex* if: 1) a simplex $\sigma \in K$ implies that all faces of σ also belong to K, and 2) for two simplices $\sigma_1, \sigma_2 \in K$, either $\sigma_1 \cap \sigma_2 \in K$ or $\sigma_1 \cap \sigma_2 = \emptyset$.

7.2.2 Power distance and weighted Voronoi diagram

The *power distance* $\pi(p, b_i)$ between a point $p \in \mathbb{R}^3$ and a ball $b_i = (p_i, r_i) \in B$ is defined as

$$\pi(p, b_i) = \|p - p_i\|^2 - r_i^2.$$

The *weighted Voronoi diagram* is an extension of the Voronoi diagram to weighted points. It is a partition of $\mathbb{R}^3$ based on proximity to input balls b_i in terms of the power distance. Points $p \in \mathbb{R}^3$ that are closer to the ball b_i compared to all other balls $b_j \in B$ $(j \neq i)$ constitute the *Voronoi region* of b_i. Points equidistant from two balls $b_i, b_j \in B$ and closer to the two balls compared to other balls constitute a *Voronoi face*. Similarly, points equidistant from three balls and fours balls constitute *Voronoi edges* and *Voronoi vertices* of the weighted Voronoi diagram, respectively. Figure 7.1(b) shows the weighted Voronoi diagram for a set of 2D weighted points or disks on the plane. Similar to the unweighted case, the Voronoi regions of the weighted Voronoi diagram are convex and linear. However, the weights may lead to a configuration where the Voronoi region of b_i is disjoint from b_i. This occurs when b_i is contained within another ball b_j. Further, the Voronoi region of b_i may even be empty.

7.2.3 Weighted Delaunay triangulation

The *weighted Delaunay triangulation* is the the dual of the weighted Voronoi diagram, see Figure 7.1(c). It is a simplicial complex consisting of simplices that are dual to the cells of the weighted Voronoi diagram. The following equivalent definition characterizes a simplex σ^d belonging to a Delaunay triangulation D [Weighted Delaunay Triangulation] A simplex $\sigma^d = (p_0^\sigma, p_1^\sigma, \ldots, p_d^\sigma)$ be-

longs to the weighted Delaunay triangulation D of B if and only if there exists a point $p \in \mathbb{R}^3$ such that

DT1: $\pi(p, b_0^\sigma) = \pi(p, b_1^\sigma) = \cdots = \pi(p, b_d^\sigma)$, and

DT2: $\pi(p, b_0^\sigma) \leq \pi(p, b_i)$ for $b_i \in B - \sigma^d$.

A point p that satisfies the above two conditions, DT1 and DT2, is called a *witness* for σ^d. We call a point that minimizes the distance $\pi(p, b_0^\sigma)$ and satisfies both conditions as the *closest witness*, denoted by p_{min}^σ. This minimum distance $\pi(p_{min}^\sigma, b_0^\sigma)$ is called the *size* of the simplex σ^d. A point that minimizes the distance $\pi(p, b_0^\sigma)$ and satisfies DT1 is called the $ortho\text{-}center$ p_{ortho}^σ of simplex σ^d. The distance $\pi(p_{ortho}^\sigma, b_0^\sigma)$ is called the $ortho\text{-}size$ of the simplex σ^d. Clearly, the $size$ of a simplex is lower bounded by its $ortho\text{-}size$. Figure 7.2 shows the two possible scenarios, namely when $ortho\text{-}size = size$ and $ortho\text{-}size < size$.

7.2.4 Alpha complex

Given a parameter $\alpha \in \mathbb{R}$, we can construct a subset of the weighted Delaunay triangulation by filtering simplices whose $size$ is less than or equal to α, see Figures 7.1(d) and 7.1(e). The resulting subset is a subcomplex of the Delaunay complex and is denoted K_α:

$$K_\alpha = \{\sigma^d \in D \quad \text{such that} \quad size(\sigma^d) \leq \alpha\}.$$

The following equivalent definition characterizes simplices of the alpha complex without explicitly referring to the Delaunay triangulation. [Alpha complex] A simplex $\sigma^d = (p_0^\sigma, p_1^\sigma, \ldots, p_d^\sigma)$ belongs to the alpha complex K_α of B if and only if there exists a point $p_{min}^\sigma \in \mathbb{R}^3$ such that the following three conditions are satisfied:

AC1: $\pi(p_{min}^\sigma, b_0^\sigma) = \pi(p_{min}^\sigma, b_1^\sigma) = \cdots = \pi(p_{min}^\sigma, b_d^\sigma)$,

AC2: $\pi(p_{min}^\sigma, b_0^\sigma) \leq \pi(p_{min}^\sigma, b_i)$ for $b_i \in B - \sigma^d$, and

AC3: $\pi(p_{min}^\sigma, b_0^\sigma) \leq \alpha$ or $size$ of σ^d is at most α.

7.3 Algorithm

We now describe an algorithm to compute the alpha complex and prove its correctness. The algorithm utilizes the characterizing conditions introduced above. It first identifies the tetrahedra that belong to the alpha complex, followed by the set of triangles, edges and vertices. Figure 7.3 illustrates the algorithm as applied to disks on the plane.

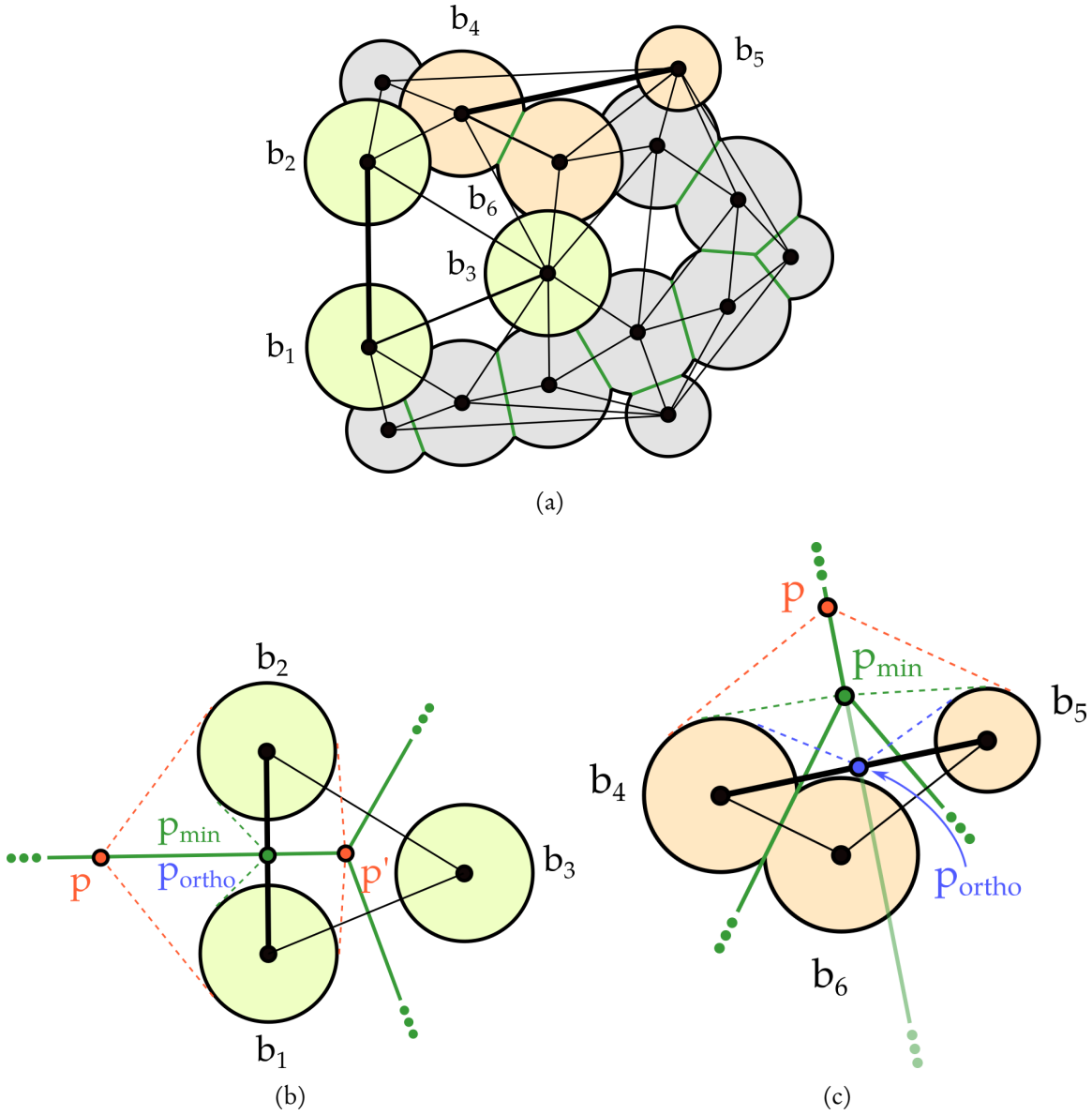


Figure 7.2: $Size$ and $ortho\text{-}size$ of a simplex. (a) A set B of weighted points. Two edges (bold) belong to the Delaunay triangulation. (b) This $size$ of edge b_1b_2 is equal to its $ortho\text{-}size$. Points p, p', p_{min} and p_{ortho} are *witnesses*. Each one is equidistant from b_1 and b_2 and farther away from other disks in B. The distance is proportional to the length of the tangent to the disk that represents the weighted point. The next closest disk from these points is b_3. In this case, p_{min} and p_{ortho} coincide and hence $size = ortho\text{-}size$. (c) b_4b_5 is also a Delaunay edge. The location of a neighboring disk b_6 could lead to a different configuration. The point p_{ortho} is closest to b_4 and b_5 among all the points that are equidistant from both. However p_{ortho} is closer to b_6 as compared to b_4 and b_5. The closest point p_{min} that satisfies DT1 and DT2 is farther away, hence $size$ of b_4b_5 is greater than its $ortho\text{-}size$.

7.3.1 Outline

The alpha complex consists of simplices of dimensions 0–3, $K_\alpha = K_\alpha^0 \cup K_\alpha^1 \cup K_\alpha^2 \cup K_\alpha^3$, where $K_\alpha^d \subset K_\alpha$ is the set of d-dimensional simplices in K_α. We initialize $K_\alpha^d = \emptyset$ and construct K_α in five steps described below:

Step 1: Compute the set of all simplices σ^d such that $ortho\text{-}size(\sigma^d) \leq \alpha$. Let this set be denoted by $\Sigma_{ortho} = \Sigma_{ortho}^0 \cup \Sigma_{ortho}^1 \cup \Sigma_{ortho}^2 \cup \Sigma_{ortho}^3$.

Step 2: For all tetrahedra $\sigma^3 \in \Sigma_{ortho}^3$, check condition AC2 using $p = p_{ortho}^\sigma$. If σ^3 satisfies AC2 then insert it into K_α^3.

Step 3: Insert all triangles that are incident on tetrahedra in K_α^3 into K_α^2. Let $\Sigma_{free}^2 = \Sigma_{ortho}^2 - Facets(K_\alpha^3)$, where $Facets(K_\alpha^3)$ denotes the set of facets of tetrahedra in K_α^3. For all triangles $\sigma^2 \in \Sigma_{free}^2$, check condition AC2 using $p = p_{ortho}^\sigma$. If σ^2 satisfies AC2 then insert it into K_α^2.

Step 4: Insert all edges incident on triangles in K_α^2 into K_α^1. Let $\Sigma_{free}^1 = \Sigma_{ortho}^1 - Facets(K_\alpha^2)$, where $Facets(K_\alpha^2)$ denotes the set of facets of triangles in K_α^2. For all edges $\sigma^1 \in \Sigma_{free}^1$, check condition AC2 using $p = p_{ortho}^\sigma$. If σ^1 satisfies AC2 then insert it into K_α^1.

Step 5: Insert all end points of edges in K_α^1 into K_α^0. Let $\Sigma_{free}^0 = \Sigma_{ortho}^0 - Facets(K_\alpha^1)$, where $Facets(K_\alpha^1)$ denotes the set of balls incident on edges in K_α^1. For all balls $b_i = (p_i, r_i) \in \Sigma_{free}^0$, check condition AC2 using $p = p_i$. If p_i satisfies AC2 then insert it into K_α^0.

Step 1 selects simplices that satisfy AC3. Step 2 recognizes tetrahedra that belong to the alpha complex by checking AC2 using $p = p_{ortho}^\sigma$. Triangle faces of these tetrahedra also belong to K_α. The other “dangling" triangles belong to K_α^2 if they satisfy AC2. Step 4 identify edges similarly. First all edge faces of triangles in K_α^2 are inserted followed by those “dangling" edges that satisfy AC2. Vertices are identified similarly in Step 5.

7.3.2 Proof of correctness

We now prove that that the algorithm described above correctly computes the alpha complex of the given set of weighted points by proving the following four claims. Each claim states that the set of simplices computed in Steps 2, 3, 4 and 5 are exactly the simplices belonging to the alpha complex.

Claim 1. *Step 2 computes K_α^3 correctly.*

Proof. We assume that the input is non-degenerate. So, for a tetrahedron σ^3, p_{ortho}^σ is the only point that satisfies condition AC1. In Step 2 of the proposed algorithm, we check if AC2 holds for p_{ortho}^σ.

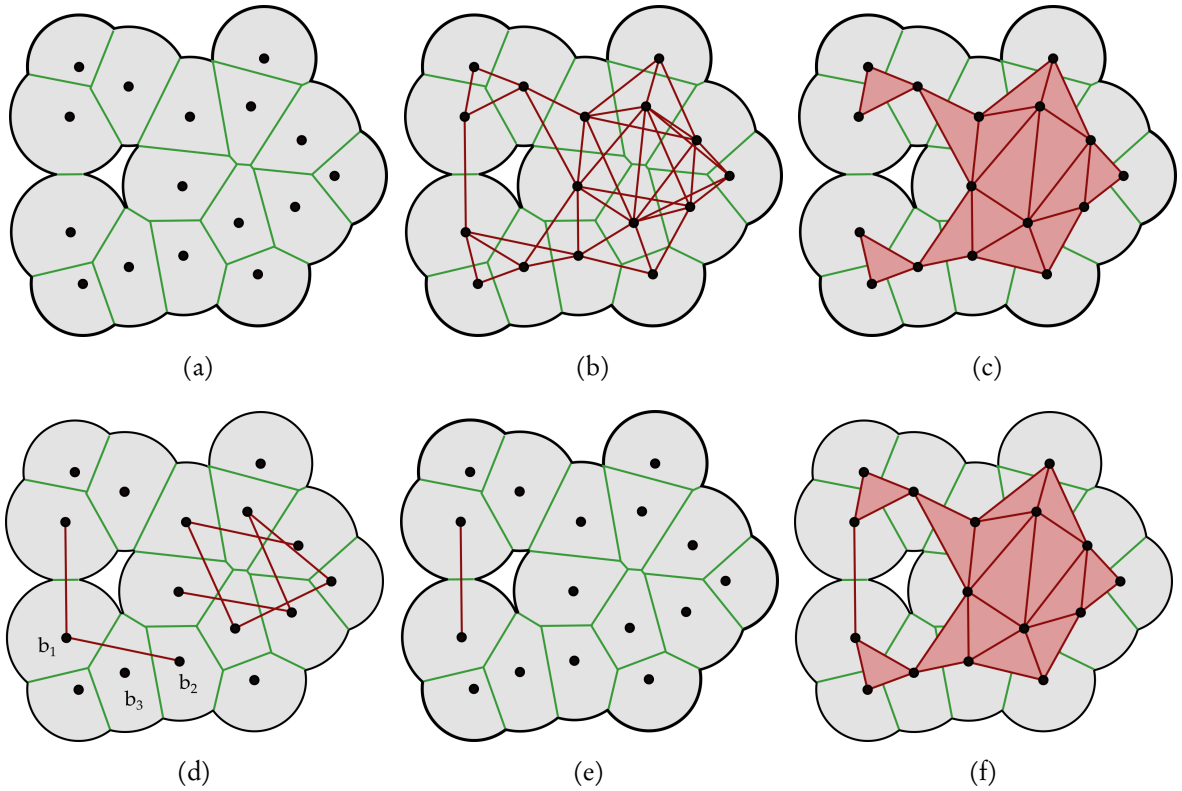


Figure 7.3: Illustration of the proposed algorithm in 2D. (a) The set of disks B grown by the parameter α. (b) First, compute the set of edges Σ^1_{ortho} whose $ortho\text{-}size \leq \alpha$ (red). The triangles Σ^2_{ortho} that satisfy this condition are also computed but they are not shown here. (c) Next, identify the triangles that satisfy AC2 (red). (d) Collect edges in Σ^1_{ortho} that are not incident on triangles in K^2_α into Σ^1_{free}. Check if these edges satisfy AC2 with $p = p^\sigma_{ortho}$. For example, the edge b_1b_2 does not satisfy this condition because b_3 is closer to p_{ortho} than b_1 and b_2. (e) Only one edge survives the AC2 check and thus belongs to K_α. (f) The alpha complex is obtained as the union of K^2_α, K^1_α and K^0_α.

If yes, then p^σ_{ortho} is a witness for σ^3, *i.e.*, $p^\sigma_{ortho} = p^\sigma_{min}$. Further, since $ortho\text{-}size(\sigma^3) \leq \alpha$ and $p^\sigma_{ortho} = p^\sigma_{min}$, we have $\text{size}(\sigma^3) \leq \alpha$ thereby satisfying AC3. Therefore, σ^3 belongs to K^3_α because it satisfies all three conditions. □

We now prove that the algorithm correctly identifies the triangles of the alpha complex.

Lemma 7.1. *A triangle $\sigma^2 \in \Sigma^2_{free}$ belongs to K^2_α if and only if it satisfies AC2 with $p = p^\sigma_{ortho}$.*

Proof. We first prove the backward implication, namely if $\sigma^2 \in \Sigma^2_{free}$ satisfies AC2 with $p = p^\sigma_{ortho}$, then $\sigma^2 \in K^2_\alpha$. Note that p^σ_{ortho} satisfies AC1 by definition. Further, it satisfies AC2 by assumption and hence $size(\sigma^2) = ortho\text{-}size(\sigma^2)$. We also have $ortho\text{-}size(\sigma^2) \leq \alpha$ because $\sigma^1 \in \Sigma^2_{free} \subseteq \Sigma^2_{ortho}$. So, $size(\sigma^2) \leq \alpha$ thereby satisfying AC3. The triangle σ^2 with $p = p^\sigma_{ortho}$ satisfies all three

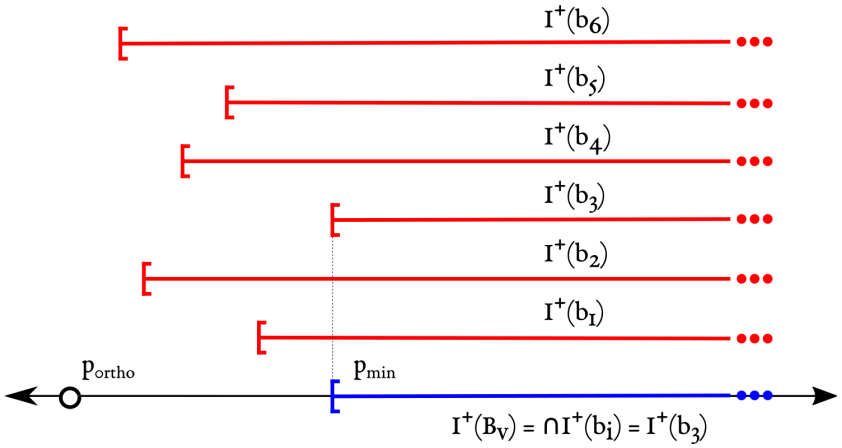


Figure 7.4: The radical axis of a triangle σ^2 is drawn such that p^σ_{ortho} is at the origin. A ball $b_i \in B - \sigma^2$ divides the radical axis into two half intervals. Points in the half interval $I^+(b_i)$ are closer to b^σ_0 as compared to b_i, *i.e.* for all $p \in I^+(b_i)$, $\pi(p, b^\sigma_0) < \pi(p, b_i)$. Consider the set B_v of balls that are closer to p^σ_{ortho} as compared to b^σ_0 So, $I^+(b_i)$ does not contain p^σ_{ortho}. The intersection of these intervals, denoted by $I^+(B_v)$, is equal to one of the intervals $I^+(b_i)$. For example, here $I^+(B_v) = I^+(b_3)$. The end point of the interval $I^+(b_3)$ is the closest witness for the tetrahedron $\sigma^2 \cup b_3$.

conditions and hence belongs to K^2_α.

We will now prove the forward implication via contradiction. Suppose there exists a triangle $\sigma^2 \in \Sigma^2_{free}$ that belongs to K^2_α but does not satisfy AC2 with $p = p^\sigma_{ortho}$. In other words, there exists a ball $b_i \in B - \sigma^2$ for which $\pi(p^\sigma_{ortho}, b_i) < \pi(p^\sigma_{ortho}, b^\sigma_0)$. Let B_v denote the set of all such balls b_i. The set of points that are equidistant from the three balls $(b^\sigma_0, b^\sigma_1, b^\sigma_2)$ corresponding to σ^2 form a line perpendicular to the plane containing σ^2 called the *radical axis*. Each ball $b_i \in B_v$ partitions the radical axis into two half-intervals based on whether the point on radical axis is closer to b_i or to b^σ_0, see Figure 7.4. Let $I^+(b_i)$ denote the half interval consisting of points that are closer to b^σ_0 compared to b_i. Let $I^+(B_v)$ denote the intersection of all such half intervals $I^+(b_i)$. We have assumed that $\sigma^2 \in K^2_\alpha$, so there must exist a closest witness p^σ_{min} and it has to lie within $I^+(B_v)$. Thus, $I^+(B_v)$ is non-empty. In fact, $I^+(B_v) = I^+(b_j)$ for some $b_j \in B_v$ and p^σ_{min} is exactly the end point of $I^+(b_j)$. This implies that p^σ_{min} is also a closest witness for the tetrahedron $\sigma^3 = (b^\sigma_0, b^\sigma_1, b^\sigma_2, b_j)$. So, σ^3 belongs to K^3_α and its size is equal to $size(\sigma^2)$. However, this means that $\sigma^2 \notin \Sigma^2_{free}$, a contradiction. So, the forward implication in the lemma is true. □

Claim 2. *Step 3 computes K^2_α correctly.*

Proof. If a simplex σ^3 belongs to K_α then naturally all of its faces also belong to K_α. The algorithm includes such triangles into K^2_α and remove them from Σ^2_{ortho} to obtain the set of free triangles Σ^2_{free} that have $ortho\text{-}size \leq \alpha$ and are not incident on any tetrahedron in K_α. It follows directly from

Lemma 7.1 that AC2 is a necessary and sufficient condition for a triangle in Σ^2_{free} to belong to K^2_α. Hence, Step 3 correctly computes the triangles belonging to K^2_α. □

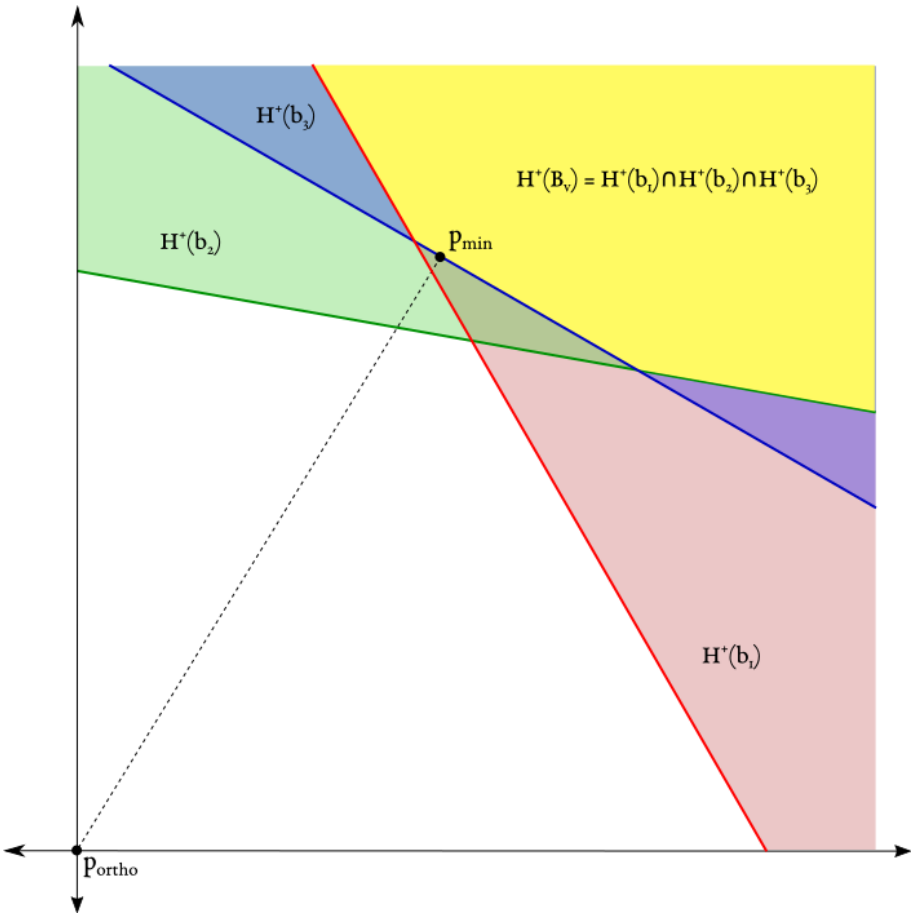


Figure 7.5: The radical plane of an edge σ^1 is drawn such that p^σ_{ortho} is at the origin. A ball $b_i \in B - \sigma^1$ divides the radical plane into two half planes. The half plane $H^+(b_i)$ consists of points that are closer to b^σ_0 as compared to b_i, *i.e.* for all $p \in H^+(b_i)$, $\pi(p, b^\sigma_0) < \pi(p, b_i)$. Let B_v denote the set of balls that are closer to p^σ_{ortho} as compared to b^σ_0. The half planes $H^+(b_i)$ do not contain p^σ_{ortho}. The intersection of these half planes, denoted by $H^+(B_v)$, is a convex region (yellow). The power distance from b^σ_0 to a point $p \in H^+(B_v)$ is minimized at a point on the boundary of the convex region $H^+(B_v)$. But the boundary of $H^+(B_v)$ is a union of line segments that bound half planes $H^+(b_i)$. Here, the point at which the distance is minimum lies on the boundary of the half plane $H^+(b_3)$. This point is the closest witness for the triangle $\sigma^1 \cup b_3$.

The above arguments need to be extended to prove that the edges of the alpha complex are also correctly identified.

Lemma 7.2. *An edge $\sigma^1 \in \Sigma^1_{free}$ belongs to K^1_α if and only if it satisfies the condition AC2 with $p = p^\sigma_{ortho}$.*

Proof. First, we assume that $\sigma^1 \in \Sigma^1_{free}$ satisfies AC2 with $p = p^\sigma_{ortho}$. The point p^σ_{ortho} satisfies AC1 by definition. Further, it satisfies AC2 by assumption and hence $size(\sigma^1) = ortho\text{-}size(\sigma^1)$. We also have $ortho\text{-}size(\sigma^1) \leq \alpha$ because $\sigma^1 \in \Sigma^1_{free} \subseteq \Sigma^1_{ortho}$. So, $size(\sigma^1) \leq \alpha$ thereby satisfying AC3. The edge σ^1 with $p = p^\sigma_{ortho}$ satisfies all three conditions and hence belongs to K^1_α.

We will prove the forward implication via contradiction. Suppose there exists an edge $\sigma^1 \in \Sigma^1_{free}$ that belongs to K^1_α but does not satisfy AC2 with $p = p^\sigma_{ortho}$. In other words, there exists a ball $b_i \in B - \sigma^1$ such that $\pi(p^\sigma_{ortho}, b_i) < \pi(p^\sigma_{ortho}, b^\sigma_0)$. Let B_v denote the set of all such balls b_i. The set of points that are equidistant from the two balls (b^σ_0, b^σ_1) corresponding to σ^1 form a plane perpendicular to the line containing σ^1 called the *radical plane*. Each ball $b_i \in B_v$ partitions the radical plane into two half-planes based on whether the point on the radical plane is closer to b_i or to b^σ_0, see Figure 7.5. Let $H^+(b_i)$ denote the half-plane consisting of points that are closer to b^σ_0 compared to b_i. Let $H^+(B_v)$ denote the intersection of all such half-planes $H^+(b_i)$. We have assumed that $\sigma^1 \in K^1_\alpha$, so there must exist a closest witness p^σ_{min} and it has to lie within $H^+(B_v)$. Thus, $H^+(B_v)$ is non-empty. In fact, p^σ_{min} lies on the envelope of $H^+(B_v)$ because it minimizes the distance to b^σ_0. Let p^σ_{min} lie on the bounding line corresponding to $H^+(b_j)$ for some $b_j \in B_v$. This implies that p^σ_{min} is also a closest witness for the triangle $\sigma^2 = (b^\sigma_0, b^\sigma_1, b_j)$. So, σ^2 belongs to K^2_α and its size is equal to $size(\sigma^1)$. However, this means that $\sigma^1 \notin \Sigma^1_{free}$, a contradiction. So, the forward implication in the lemma is also true. □

Claim 3. *Step 4 computes K^1_α correctly.*

Proof. All edge faces of triangles in K^2_α naturally belong to K^1_α. Step 4 inserts all edges incident on triangles in K^2_α into K^1_α as valid edges and removes them from Σ^1_{ortho} to obtain the set of free edges Σ^1_{free}. It follows directly from Lemma 7.2 that AC2 is a necessary and sufficient condition for an edge $\sigma^1 \in \Sigma^1_{free}$ to belong to K^1_α. Therefore, Step 4 correctly computes the edges belonging to K^1_α. □

Claim 4. Step 5 computes K^0_α correctly.

Proof. All vertices incident on K^1_α naturally belong to K^1_α. Step 5 inserts all such vertices in K^0_α as valid vertices and removes them from Σ^0_{ortho} to obtain the set of free vertices Σ^0_{free}. Next, the vertices in Σ^0_{free} for which the center of the ball $b_i = (p_i, r_i)$ satisfies AC2 are also inserted into K^0_α. Clearly, these vertices also satisfy AC3 because they belong to Σ^0_{ortho}. The condition AC1 is not relevant for 0-dimensional simplices. Therefore, these vertices clearly belong to the alpha complex. Similar to Lemmas 7.1 and 7.2, it is easy to prove that checking for AC2 for $p = p_i$ is necessary and sufficient condition to decide whether a vertex in Σ^0_{free} belongs to the alpha complex or not. That is, it is possible to show that vertices in alpha complex that have non-empty Voronoi regions but

do not satisfy AC2 for $p = p_i$ would be incident on some edge in K^1_α, and therefore must have been already detected by Step 4 and hence can not belong to Σ^0_{free}. Therefore, Step 5 correctly computes the vertices belonging to K^0_α. □

7.4 Parallel Algorithm for Biomolecules

Although the algorithm as described above is provably correct, a straight forward implementation will be extremely inefficient with a worst case running time of $O(n^5)$, where n is the number of weighted points in B. This is because Step 1 requires $O(n^4)$ time to generate all possible tetrahedra. In later steps, we need $O(n)$ effort per simplex to check AC2. However, the input corresponds to atoms in a biomolecule. We show how certain properties of biomolecules can be leveraged to develop a fast parallel implementation.

7.4.1 Biomolecular data characteristics

Atoms in a biomolecule are well distributed. The following three properties of biomolecules are most relevant:

- The radius of an atom is bounded and very small compared to the size of the biomolecule. The typical radius of an atom in a protein molecule ranges between 1Å to 2Å [9].
- There is a lower bound on the distance between the centres of two atoms. This is called the *van der Waals contact distance*, beyond which the two atoms start repelling each other. In the case of atoms in protein molecules, this distance is at least 1Å. This property together with the upper bound on atomic radii ensures that no atom is completely contained inside another. This means that the weighted Voronoi regions corresponding to the atoms in a biomolecule can be always be assumed to be non-empty.
- Structural biologists are interested in small values of α. The two crucial values are 0Å and 1.4Å. The former corresponds to using van der Waals radius and the latter corresponds to the radius of water molecule, which acts as the solvent.

In the light of the above three properties, we can say that the number of simplices of the alpha complex that are incident on a weighted point (atom) is independent of the total number of input atoms and is bounded by a constant [48].

7.4.2 Acceleration data structure

The algorithm will benefit from an efficient method for accessing points of B that belong to a local neighborhood of a given weighted point. We store the weighted points in a grid-based data

structure. Let r_{max} denote the radius of the largest atom and assume that the value of the parameter α is available as input. First, we construct a grid with cells of size $\sqrt{r_{max}^2 + \alpha}$ and then bin the input atoms into the grid cells. In our implementation, we do not store the grid explicitly because it may contain several empty cells. Instead, we compute the cell index for each input atom and sort the list of atoms by cell index to ensure that atoms that belong to a particular cell are stored at consecutive locations. The cell index is determined based on a row major or column major order. Alternatively, a space-filling curves like the Hilbert curve could also be used to order the cells.

After the atoms are stored in grid cells, the alpha complex is computed in two stages. In the first stage, we employ a bottom up approach to obtain a conservative estimate of the edges, triangles, and tetrahedra belonging to the alpha complex. The false positives from the first stage are removed in a subsequent pruning stage resulting in the correct alpha complex.

7.4.3 Potential simplices

The first stage essentially corresponds to Step 1 of the algorithm described in the previous section. We compute the set Σ_{ortho} of potential simplices for which $ortho\text{-}size(\sigma^d) \leq \alpha$. However, for efficiency reasons we process the simplices in the order of increasing dimension. First, we identify edges that satisfy the AC3 condition. Given the size of the grid cell, end points of edges that satisfy the condition either lie within the same grid cell or in adjacent cells. So, the grid data structure substantially reduces the time required to compute the list of potential edges Σ^1_{ortho}. Beginning from this set of edges, we construct the set of all possible triangles and retain the triangles whose $ortho\text{-}size$ is no greater than α, resulting in the set Σ^2_{ortho}. Finally, we use the triangles in Σ^2_{ortho} to construct the list of tetrahedra that satisfy the $ortho\text{-}size \leq \alpha$ condition. The above procedure works because the $ortho\text{-}size$ of a simplex is always greater than or equal to the $ortho\text{-}size$ of its faces. The set of simplices identified in this stage contains all simplices of the alpha complex. False positives are pruned in the the second stage described below.

7.4.4 Pruning

The second stage corresponds to Steps 2-5 of the algorithm and processes the simplices in the decreasing order of dimension. This stage checks the characterizing condition AC2 to prune Σ_{ortho} into K_α. The tetrahedra are processed by checking if any of the input balls are closer to the $ortho\text{-}center$ than the balls incident on the tetrahedron. If yes, the tetrahedron is pruned away. Else, the tetrahedron is recognized as belonging to the alpha complex and inserted into K^3_α. Triangles incident on these tetrahedra also belong to the alpha complex and are inserted into K^2_α after they are removed from the list of potential triangles Σ^2_{ortho}. Next, the triangles in Σ^2_{ortho} are processed by checking if they satisfy AC2. If yes, they are inserted into K^2_α. Else, they are pruned away. All edges incident

on triangles belonging to K_α^2 are inserted into K_α^1 and removed from the set Σ_{ortho}^1. Next, the edges in Σ_{ortho}^1 are processed by checking if they satisfy AC2. Edges that satisfy AC2 are inserted into K_α^1 and the others are pruned away. All the vertices in Σ_{ortho}^0 are directly inserted into K_α^0 without AC2 check because for biomolecular data we assume that Voronoi regions of all the atoms are non-empty. The check for condition AC2 for each simplex is again made efficient by the use of the grid data structure. Atoms that may violate AC2 lie within the same cell as that containing the $ortho\text{-}center$ or within the adjacent cell. Atoms that lie within other cells may be safely ignored.

7.4.5 CUDA implementation

We use the CUDA framework [1] and the `thrust` library [2] within CUDA to develop a parallel implementation of the algorithm that executes on the many cores of the GPU. The grid computation is implemented as a CUDA kernel where each atom is processed in parallel. The potential simplices computation and pruning stages are broken down into multiple CUDA kernels and parallelized differently in order to increase efficiency. We now describe the parallelization strategy in brief. For computing the set of potential edges Σ_{ortho}^1, the initial enumeration of all possible edges is parallelized per atom where atom indices are used to ensure that no duplicate edges are generated. Subsequently, the $ortho\text{-}size$ condition is checked for the edges in parallel. Similarly, for computing potential triangles Σ_{ortho}^2, the initial enumeration of all possible triangles is parallelized per atom, while the $ortho\text{-}size$ condition is checked within a separate kernel and parallelized per triangle. The same strategy is used for computing the set of potential tetrahedra Σ_{ortho}^3. The pruning stage is parallelized per tetrahedron, triangle, and edge as required.

7.4.6 Handling large data sizes

Typical protein structures consist of upto 100,000 atoms. Our implementation can handle datasets of this size easily for reasonable values of α. However, the size of datasets is ever increasing. Protein complexes that are available nowadays may consist of millions of atoms, necessitating smart management of GPU memory while handling such data sets.

We propose two strategies and implement one of them. The first strategy is to partition the grid by constructing an octree data structure and choosing an appropriate level in the octree to create partitions. Each partition together with its border cells can be processed independently of other partitions. So, we can copy one partition and its border to the GPU memory, compute its alpha complex, and copy the results back from GPU to CPU memory. After all the partitions are processed, the list of simplices can be concatenated followed by duplicate removal to generate the final alpha complex.

The second strategy is to partition the sorted list of atoms into equal sized *chunks* and to process

each chunk independently. Here, we assume that the complete list of atoms together with the grid data structure fits in the GPU memory. This is a reasonable assumption considering that datasets containing several million atoms can easily fit on modern GPUs, which typically have at least 2GB video memory. Also, the main difficulty in handling large protein structures is managing the large lists of simplices generated within the intermediate steps of the algorithm, when compared to handling the input list of atoms or the output list of simplices. We compute the alpha complex by executing the algorithm in multiple passes. Each pass computes the alpha complex for a single chunk and copies it back to the CPU memory. We have implemented this second strategy and can handle data sizes of upto 16 million atoms on a GPU with 2GB of memory. Results are reported in the next section.

7.5 Experimental Results

We now present results of computational experiments that demonstrate that the parallel algorithm is fast in practice and significantly better than the state-of-the-art. We also performed runtime profiling to better understand the bottlenecks and effect of the parameter α on the runtime. All experiments, unless stated otherwise, were performed on a Linux system with an nVidia GTX 660 Ti graphics card running CUDA 8.0 and a 2.0GHz Intel Xeon octa core processor with 16 GB of main memory. The default number of threads per block was set at 512 for all the CUDA kernels.

Mach and Koehl describe two techniques for computing alpha complex of biomolecules called *AlphaVol* and *UnionBall* in their paper [72]. Both approaches construct the weighted Delaunay triangulation of input atoms first followed by a filtering step to obtain the alpha complex. *UnionBall* is the state-of-the-art technique for alpha complex computation for biomolecules on multi-core CPU. It uses heuristics and optimizations specific to biomolecular data to improve upon *AlphaVol*. For biomolecules containing 5 million atoms, *AlphaVol* takes approximately 8600 seconds for computing the alpha complex, while *UnionBall* takes approximately 150 seconds. Our method computes the alpha complex in less than 3 seconds for similar sized data, see Table 7.1.

7.5.1 Comparison with *gReg3D*

We are not aware of any algorithm that can compute the alpha complex directly without first constructing the complete Delaunay triangulation. In order to compare the performance, we chose the state-of-the-art parallel algorithm for computing the weighted Delaunay triangulation in 3D, *gReg3D* [15]. The CUDA implementation of *gReg3D* is available in the public domain. Table 7.1 compares the running times of our proposed algorithm with that of *gReg3D* for 12 different biomolecules at $\alpha = 0$ and $\alpha = 1$. As evident from the table, we consistently observe significant speedup over *gReg3D*. The observed speedup is as high as $22\times$ for the biomolecule 1X9P at $\alpha = 0$,

Table 7.1: Runtime comparison of the proposed algorithm with *gReg3D* on an nVidia GTX 660 Ti graphics card. Timings are reported in milliseconds. %Simplex refers to the size of the alpha complex as a percentage of the size of the weighted Delaunay triangulation. The last column shows the speedup in runtime of our algorithm over *gReg3D*. '*' indicates the data was partitioned and processed in chunks. '–' indicates that the code could not execute due to insufficient memory.

α	PDB id	#Atoms	K_α		*gReg3D*		%Simplex	Speed up
			#Simplices	Time(ms)	#Simplices	Time(ms)		
0.0	1GRM	260	932	13	6295	117	14.8	**9.0**
	1U71	1505	5696	13	40878	115	13.9	**11.1**
	3N0H	1509	5739	14	41244	137	13.9	**10.0**
	4HHB	4384	38796	29	150141	193	25.8	**6.6**
	2J1N	8142	29642	18	227719	229	13.0	**12.7**
	1K4C	16068	62851	27	446383	347	14.1	**12.9**
	2OAU	16647	123175	56	466586	344	26.4	**6.2**
	1AON	58674	262244	65	1650841	879	15.9	**13.5**
	1X9P*	217920	924086	113	6142811	2555	15.0	**22.6**
	1IHM*	677040	2713083	277	–	–	–	–
	4CWU*	5905140	23450403	2709	–	–	–	–
	3IYN*	5975700	24188892	2874	–	–	–	–
1.0	1GRM	260	1598	15	6295	117	25.4	**7.9**
	1U71	1505	10828	17	40878	115	26.5	**8.5**
	3N0H	1509	10965	30	41244	137	26.6	**4.6**
	4HHB	4384	65987	86	150141	193	44.0	**2.2**
	2J1N	8142	58205	30	227719	229	25.6	**7.6**
	1K4C	16068	118467	52	446383	347	26.5	**6.7**
	2OAU	16647	199101	159	466586	344	42.7	**2.2**
	1AON	58674	495683	160	1650841	879	30.0	**5.5**
	1X9P*	217920	1653778	196	6142811	2555	26.9	**13.0**
	1IHM*	677040	5058507	605	–	–	–	–
	4CWU*	5905140	44411353	5118	–	–	–	–
	3IYN*	5975700	45790463	5501	–	–	–	–

one of the largest molecules in our dataset. Clearly, the speedup goes down for $\alpha = 1$ when compared to $\alpha = 0$ because of the increased number of simplices in the output alpha complex. We also report the number of simplices in the alpha complex compared to the total number of simplices in the Delaunay triangulation under the column '%Simplex'. This makes it clear why the speedup goes down as α is increased from 0 to 1. For example, for the protein 1AON, the fraction of alpha complex simplices increases from 15.9% to 30% as α is increased from 0 to 1. Correspondingly, the speedup decreases from 13.5× to 5.5×.

We repeated the experiment on a MS Windows system with an nVidia GTX 980 Ti card running

CUDA 8.0 and observed similar speedups. However, the individual runtimes both for our algorithm and for *gReg3D* were higher on the GTX 980 Ti. We believe that the reason for this increased runtime is that the MS Windows system was utilizing the GPU resources for its various GUI tasks whereas the Linux system did not require as many GPU cycles.

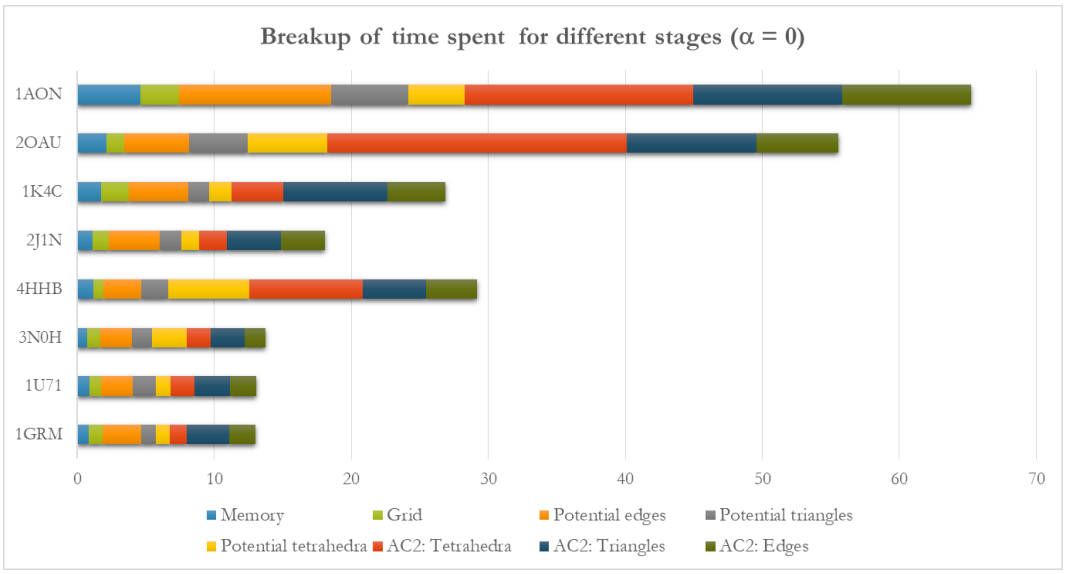
The starred entries in Table 7.1 are results for execution using the data partitioning approach. This is necessitated because these four large molecules generate large intermediate simplex lists that can not fit into the GPU memory if all the atoms in the molecule are processed at once. We observe that *gReg3D* is able to successfully compute the Delaunay complex for only one out of these four large molecules and runs out of GPU memory for the remaining three molecules.

7.5.2 Runtime profiling

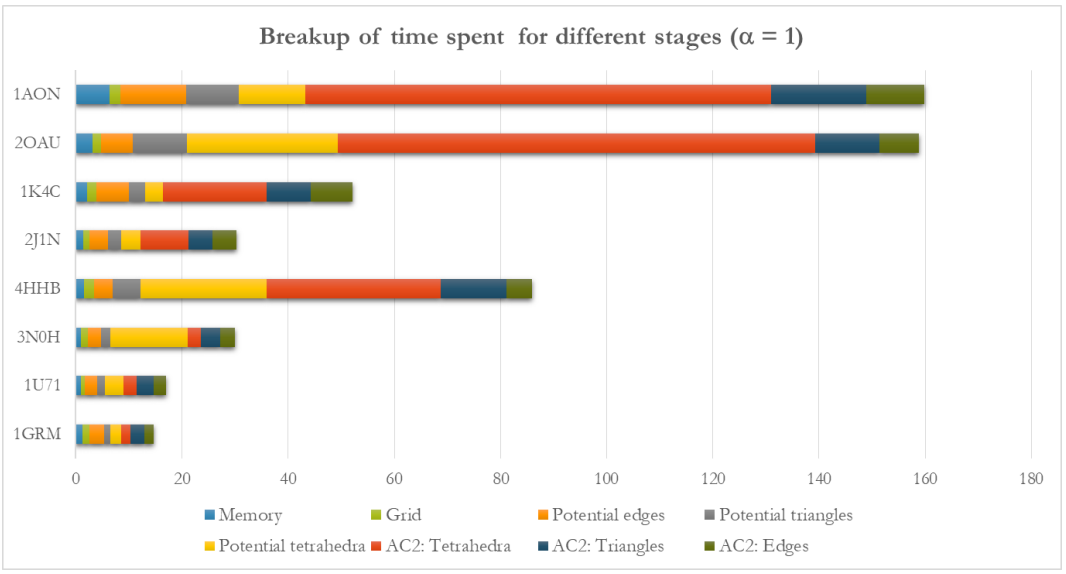
The two stages of our parallel algorithm (potential simplices and pruning) are further divided into 3 steps each, corresponding to the computation of edges, triangles, and tetrahedra respectively. A grid computation step precedes the two stages. We study the computation effort for each of these seven steps of the algorithm. We also report the time spent in memory transfers from CPU to GPU and vice-versa. Thus, we report the split up of the total runtime into eight categories namely, 'Memory transfer', 'Grid computation', 'Potential edges', 'Potential triangles', 'Potential tetrahedra', 'AC2

Table 7.2: Time spent within different steps of the algorithm. Timings are reported in milliseconds for memory transfer, grid computation, computing potential simplices, and pruning. The last column shows the total time taken for all steps.

α	PDB id	#Atoms	#Simplices	Memory	Grid	Potential Simplices			Pruning			Total
						Edges	Tris	Tets	Tets	Tris	Edges	time
0.0	1GRM	260	932	0.8	1.0	2.8	1.1	1.0	1.2	3.1	1.9	13.0
	1U71	1505	5696	0.9	0.8	2.3	1.7	1.0	1.8	2.6	1.9	13.0
	3N0H	1509	5739	0.7	0.9	2.3	1.4	2.5	1.8	2.5	1.5	13.7
	4HHB	4384	38796	1.1	0.8	2.7	2.0	6.0	8.3	4.6	3.7	29.2
	2J1N	8142	29642	1.1	1.2	3.7	1.6	1.3	2.0	3.9	3.2	18.1
	1K4C	16068	62851	1.7	2.0	4.3	1.5	1.6	3.8	7.6	4.3	26.9
	2OAU	16647	123175	2.1	1.3	4.7	4.3	5.8	21.9	9.5	6.0	55.5
	1AON	58674	262244	4.6	2.8	11.1	5.6	4.1	16.7	10.9	9.4	65.2
1.0	1GRM	260	1598	1.2	1.4	2.7	1.2	2.0	1.8	2.7	1.8	14.7
	1U71	1505	10828	0.9	0.8	2.3	1.5	3.4	2.5	3.2	2.5	17.1
	3N0H	1509	10965	0.9	1.4	2.4	1.7	14.6	2.6	3.5	2.9	30.0
	4HHB	4384	65987	1.5	1.9	3.4	5.4	23.7	32.8	12.4	4.8	86.0
	2J1N	8142	58205	1.4	1.1	3.4	2.5	3.6	9.0	4.6	4.5	30.3
	1K4C	16068	118467	2.1	1.8	6.1	3.0	3.4	19.5	8.3	7.9	52.2
	2OAU	16647	199101	3.0	1.7	6.0	10.2	28.3	90.0	12.1	7.5	158.9
	1AON	58674	495683	6.3	2.0	12.4	9.9	12.5	87.9	17.9	11.0	159.9



(a) $\alpha = 0.0$

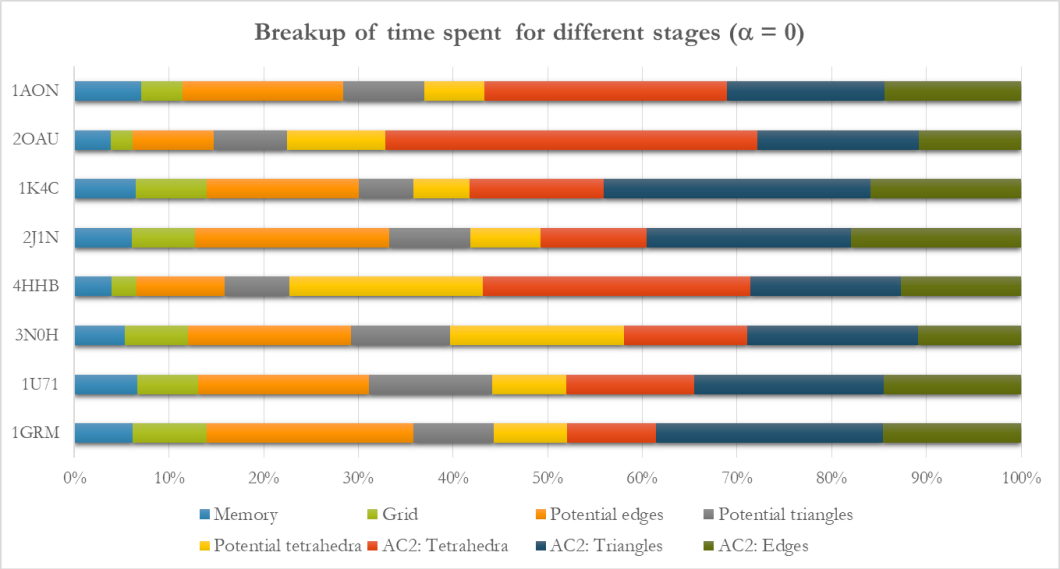


(b) $\alpha = 1.0$

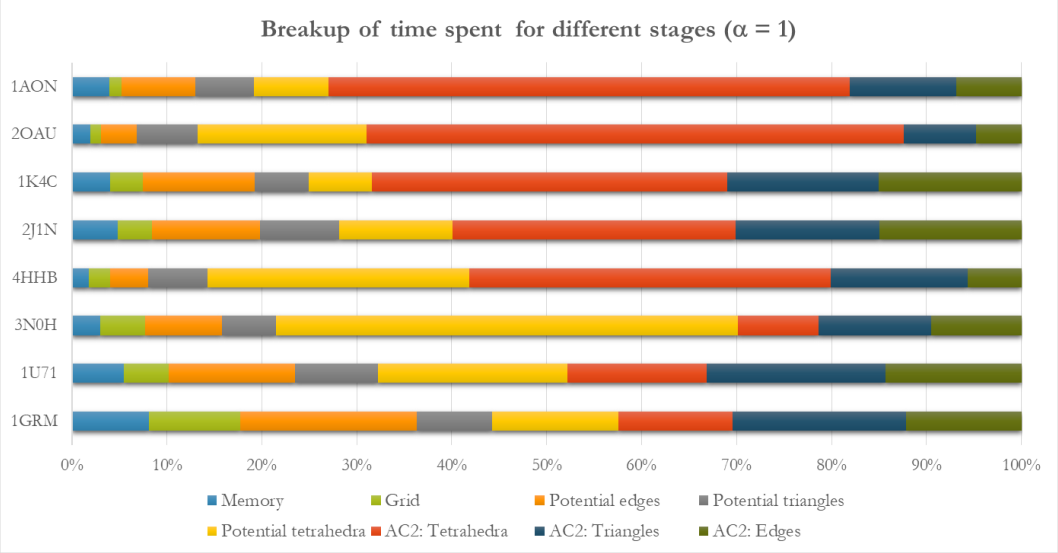
Figure 7.6: Time spent for different steps of the algorithm.

tetrahedra', 'AC2 triangles' and 'AC2 edges'.

Table 7.2 summarizes the observed split up of runtime for 8 different biomolecules. Figure 7.6 shows the actual time spent for different steps and Figure 7.7 shows relative time spent for each step. From these figures, it is clear that the pruning stage consumes the maximum amount of time. The pruning stage involves checking the neighboring balls for violations of the AC2 condition for each simplex. Specially, the tetrahedra pruning step (red) takes approximately 25% of the total time



(a) $\alpha = 0.0$



(b) $\alpha = 1.0$

Figure 7.7: Proportion of time spent for different steps during the execution of the algorithm. The pruning stage (AC2 tetrahedra, triangles and edges checks) takes significantly more effort compared to other steps. Also, the time spent for this step increases with α.

required for alpha complex computation.

We performed additional experiments to determine the average split up over multiple runs. We computed the relative time spent for each step for different values of α between 0.0 and 2.0. These observations are reported in Figure 7.8. It is clear that the memory transfers and grid computation combined do not take more than 10% of the total time. The potential simplices estimation stage

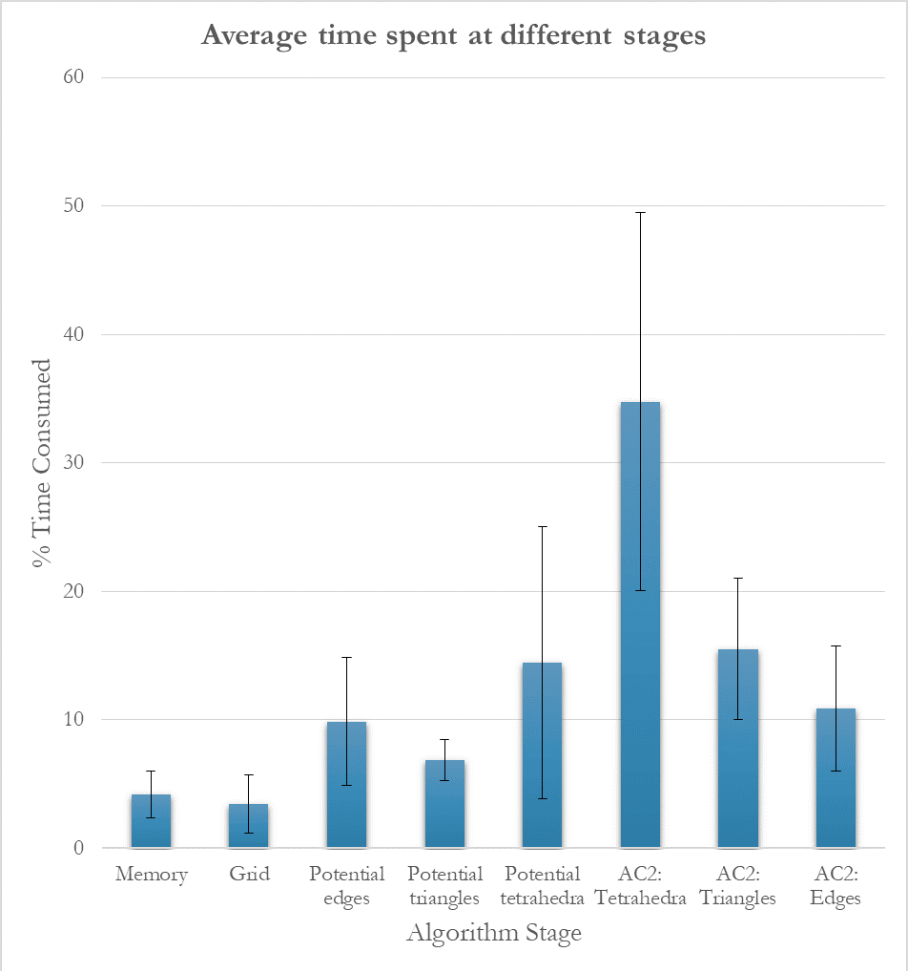
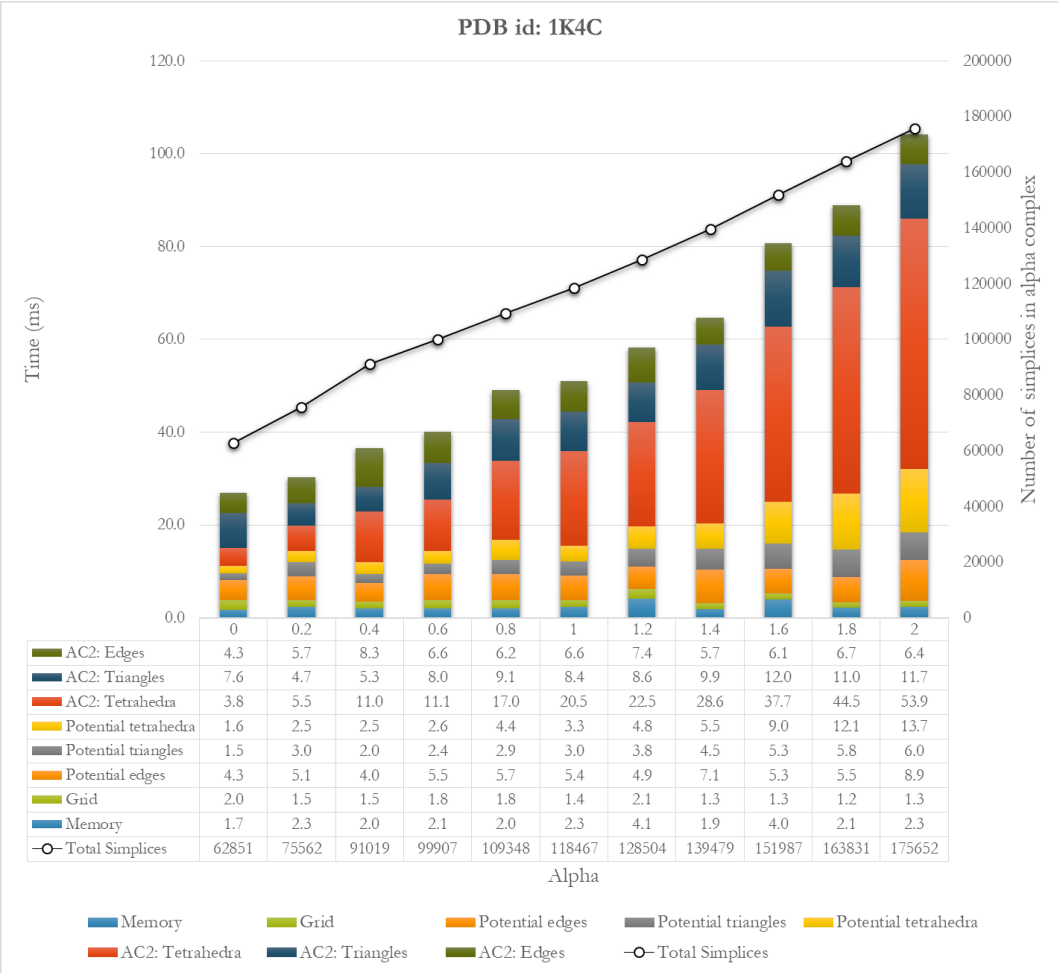


Figure 7.8: Split up of time spent at different steps. Average proportion of effort spent at different steps of computation was obtained after executing the algorithm for all the molecules in our dataset at various values of α varying from 0 to 2.0. As evident from the error bars, there is significant variability. But, in general the pruning stage of the algorithm, specially the tetrahedra computation step takes the maximum time.

consumes 30% of the time. However, the pruning stage is most expensive, taking up roughly 60% of the computation time. Pruning tetrahedra step takes up 35% of the time on average. This suggests that this step should be the focus of the optimization efforts in future. It should be noted that proportion of time spent for each step depends on the distribution of atoms in the biomolecule as well as the value of α. This explains the significant deviation from the averages as shown by the error bars.

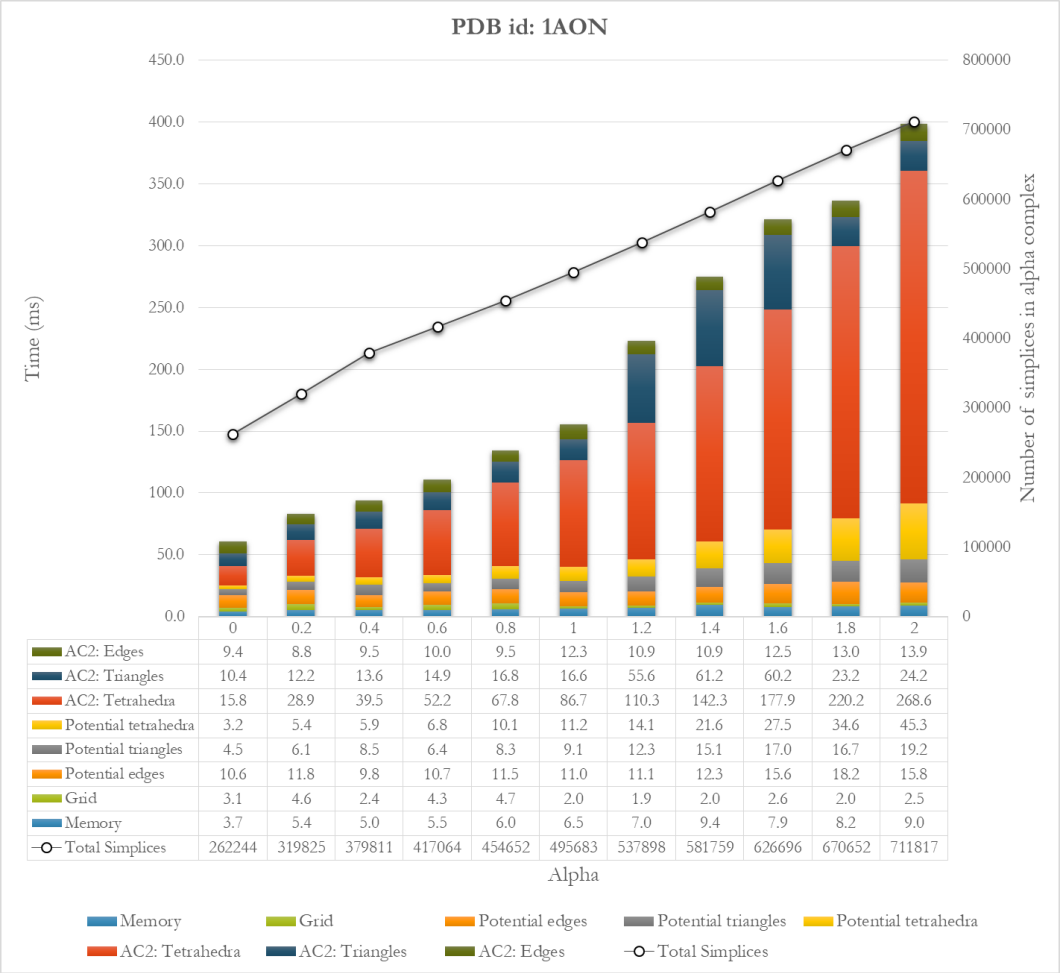
7.5.3 Effect of the value of α

We also performed experiments to observe the runtime performance as the value of α is varied between 0.0 and 2.0. Figures 7.9 and 7.10 show the results for the proteins 1K4C and 1AON, respectively. We also show how the number of simplices in the computed alpha complex increases as the value of α is increased. The runtime and total number of simplices follow a near-linear trend. However, increase in time required for pruning, especially for pruning the tetrahedra, is greater than time required for other steps of the algorithm. Note that although both graphs appear linear, this is not guaranteed behavior for other input. The scaling behavior depends on the distribution



Alpha	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
AC2: Edges	4.3	5.7	8.3	6.6	6.2	6.6	7.4	5.7	6.1	6.7	6.4
AC2: Triangles	7.6	4.7	5.3	8.0	9.1	8.4	8.6	9.9	12.0	11.0	11.7
AC2: Tetrahedra	3.8	5.5	11.0	11.1	17.0	20.5	22.5	28.6	37.7	44.5	53.9
Potential tetrahedra	1.6	2.5	2.5	2.6	4.4	3.3	4.8	5.5	9.0	12.1	13.7
Potential triangles	1.5	3.0	2.0	2.4	2.9	3.0	3.8	4.5	5.3	5.8	6.0
Potential edges	4.3	5.1	4.0	5.5	5.7	5.4	4.9	7.1	5.3	5.5	8.9
Grid	2.0	1.5	1.5	1.8	1.8	1.4	2.1	1.3	1.3	1.2	1.3
Memory	1.7	2.3	2.0	2.1	2.0	2.3	4.1	1.9	4.0	2.1	2.3
Total Simplices	62851	75562	91019	99907	109348	118467	128504	139479	151987	163831	175652

Figure 7.9: Running time for varying values of α for 1K4C. The number of simplices in the output alpha complex is also shown (black line). The number of simplices increases almost linearly with α as expected from the distribution of atoms in typical biomolecules. The running time also increases almost linearly with α for this molecule. Also, the fraction of time spent for tetrahedra computation step (red) increases with α.



Alpha	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
AC2: Edges	9.4	8.8	9.5	10.0	9.5	12.3	10.9	10.9	12.5	13.0	13.9
AC2: Triangles	10.4	12.2	13.6	14.9	16.8	16.6	55.6	61.2	60.2	23.2	24.2
AC2: Tetrahedra	15.8	28.9	39.5	52.2	67.8	86.7	110.3	142.3	177.9	220.2	268.6
Potential tetrahedra	3.2	5.4	5.9	6.8	10.1	11.2	14.1	21.6	27.5	34.6	45.3
Potential triangles	4.5	6.1	8.5	6.4	8.3	9.1	12.3	15.1	17.0	16.7	19.2
Potential edges	10.6	11.8	9.8	10.7	11.5	11.0	11.1	12.3	15.6	18.2	15.8
Grid	3.1	4.6	2.4	4.3	4.7	2.0	1.9	2.0	2.6	2.0	2.5
Memory	3.7	5.4	5.0	5.5	6.0	6.5	7.0	9.4	7.9	8.2	9.0
Total Simplices	262244	319825	379811	417064	454652	495683	537898	581759	626696	670652	711817

Figure 7.10: Running time for varying values of α for 1AON. The number of simplices in the output alpha complex is also shown (black line). The running time increases almost linearly with α for this molecule.

of the atoms in the molecule and on the range of α values for which the experiment is conducted.

7.5.4 Numerical issues

The proposed algorithm requires computation of *ortho-size* for each simplex, which in turn requires solving systems of linear equations. These computations require higher precision than is available on the GPU. So, the results may contain numerical errors. These numerical errors ultimately manifest as misclassification of a simplex as belonging to K_α or not. We performed extensive experimentation and observed that the alpha complex computed is correct in several cases. In cases where the results are not correct, the number of false positives and negatives (extra or missing simplices) is extremely small as compared to the number of simplices in the alpha complex. We observed a worst case error rate of 0.001 in our experiments, see Table 7.3. This error rate is tolerable

Table 7.3: Incorrectly identified simplices of the alpha complex.

α	PDB id	#Atoms	#Simplices	#Misclassified Simplices				Error rate
				Edges	Triangles	Tetrahedra	Total	
0.0	1GRM	260	932	0	0	0	0	0.0000
	1U71	1505	5696	0	0	0	0	0.0000
	3N0H	1509	5739	0	0	0	0	0.0000
	4HHB	4384	38796	0	0	0	0	0.0000
	2J1N	8142	29642	0	0	0	0	0.0000
	1K4C	16068	62851	15	33	16	64	0.0010
	2OAU	16647	123175	12	21	5	38	0.0003
	1AON	58674	262244	22	39	21	82	0.0003
1.0	1GRM	260	1598	0	0	0	0	0.0000
	1U71	1505	10828	0	0	0	0	0.0000
	3N0H	1509	10965	0	0	0	0	0.0000
	4HHB	4384	65987	0	0	0	0	0.0000
	2J1N	8142	58205	0	0	0	0	0.0000
	1K4C	16068	118467	20	34	14	68	0.0006
	2OAU	16647	199101	10	22	10	42	0.0002
	1AON	58674	495683	10	26	21	57	0.0001

for several applications. If exact computation is required, we could use a tolerance threshold to tag some simplices as requiring further checks, which can in turn be performed on the CPU using a multi-precision library. Our implementation can be easily extended to use such a hybrid strategy. We plan to implement this in future.

7.6 Conclusions

We proposed a novel parallel algorithm to compute the alpha complex for biomolecular data that does not require prior computation of the complete Delaunay triangulation. The novel characterization of simplices that belong to the alpha complex may be of independent interest. The algorithm was implemented using CUDA, which exploits the characteristics of the atom distribution in biomolecules to achieve speedups of upto $22\times$ compared to the the state-of-the-art parallel algorithm for computing the weighted Delaunay triangulation and upto $50\times$ speedup over the state-of-the-art implementation that is optimized for biomolecules. In future work, we plan to further improve the runtime efficiency of the parallel implementation and to resolve the numerical issues using real arithmetic.

Chapter 8

Conclusions

In recent years, techniques from the field of scientific visualization and computational geometry have increasingly found application in the study of biomolecules, specially in understanding their structure and their interaction with other molecules. This thesis is a contribution to this area of application of visualization. In this thesis, we addressed some of the challenges from the end-user perspective and some problems from a purely computational perspective. We described two new tools for the extraction and visualization of channel and cavity structures in a biomolecule aimed at the end-users, the biologists. In addition, we proposed GPU based efficient algorithms for the computation of alpha complex and discrete Voronoi diagrams. Both of these geometric structures are widely used in the study of bimolecular structure.

We described a new method for the extraction, visualization, and visual exploration of channels in biomolecules through a software tool called ChExVis. For extraction of robust cavities in uncertain data, we described a novel method of connecting molecular cavities under different optimization criteria. A web server tool called RobustCavities was designed to facilitate reliable extraction of cavities in proteins. In addition to these tools aimed at biologists, we looked at computational problems of fast computation of discrete Voronoi diagrams and alpha complex, both of which find application in the computation of biomolecular channels and cavities. Using CUDA based parallel implementation we were able to achieve significant speed-ups over the state-of-the-art methods.

Although the focus of the thesis was biomolecules, we believe the proposed techniques could be useful in other application areas as well. For instance, channel extraction technique can be used for study of porous structures in solids. Similarly, alpha complex will be useful in study of any point cloud dataset.

In future, one of the major challenges is to develop better visualization techniques for molecular dynamics simulation data, which is increasingly becoming more popular than study of static

biomolecular structure. Another major challenge is handling large protein complexes which requires time and memory efficient algorithms. Thirdly, it is important to address the problem of channel and cavity extraction in uncertain data based on sound theoretical foundations. Although we proposed solutions via connecting cavities for this problem, we believe this problem needs further exploration. We hope that this thesis will prove to be helpful in addressing these open problems.

Bibliography

[1] CUDA Zone. https://developer.nvidia.com/cuda-zone, 2017. [Online; accessed 19-December-2017]. 116

[2] Thrust. https://developer.nvidia.com/thrust, 2017. [Online; accessed 19-December-2017]. 116

[3] H. Ashkenazy, E. Erez, E. Martz, T. Pupko, and N. Ben-Tal. ConSurf 2010: calculating evolutionary conservation in sequence and structure of proteins and nucleic acids. *Nucleic Acids Res.*, 38(Web Server issue):W529–533, Jul 2010. [PubMed:20478830] [PubMed Central:PMC2896094] [doi:10.1093/nar/gkq399]. 22, 36

[4] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013. ISBN 978-981-4447-63-8. 10, 82, 103, 105

[5] Aurenhammer, Franz. Voronoi Diagrams – A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991. 82

[6] N. A. Baker, D. Sept, S. Joseph, M. J. Holst, and J. A. McCammon. Electrostatics of nanosystems: application to microtubules and the ribosome. *Proc. Natl. Acad. Sci. U.S.A.*, 98(18):10037–10041, Aug 2001. [PubMed:11517324] [PubMed Central:PMC56910] [doi:10.1073/pnas.181342398]. 21

[7] 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. 49

[8] N. Bocquet, H. Nury, M. Baaden, C. Le Poupon, J. P. Changeux, M. Delarue, and P. J. Corringer. X-ray structure of a pentameric ligand-gated ion channel in an apparently open conformation. *Nature*, 457(7225):111–114, Jan 2009. [PubMed:18987633] [doi:10.1038/nature07462]. 29, 34

[9] A Bondi. van der waals volumes and radii. *The Journal of physical chemistry*, 68(3):441–451, 1964. [doi:10.1021/j100785a001]. 114

[10] C. Bossa, M. Anselmi, D. Roccatano, A. Amadei, B. Vallone, M. Brunori, and A. Di Nola. Extended molecular dynamics simulation of the carbon monoxide migration in sperm whale myoglobin. *Biophys. J.*, 86(6):3855–3862, Jun 2004. [PubMed:15189882] [PubMed Central:PMC1304287] [doi:10.1529/biophysj.103.037432]. 65, 67, 81

[11] Adrian Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981. 103

[12] J. Brezovsky, E. Chovancova, A. Gora, A. Pavelka, L. Biedermannova, and J. Damborsky. Software tools for identification, visualization and analysis of protein tunnels and channels. *Biotechnol. Adv.*, 31(1):38–49, 2013. [PubMed:22349130] [doi:10.1016/j.biotechadv.2012.02.002]. 14

[13] M. Brunori and Q. H. Gibson. Cavities and packing defects in the structural dynamics of myoglobin. *EMBO Rep.*, 2(8):674–679, Aug 2001. [PubMed:11493595] [PubMed Central:PMC1083996] [doi:10.1093/embo-reports/kve159]. 64, 79, 80

[14] M. Brunori, B. Vallone, F. Cutruzzola, C. Travaglini-Allocatelli, J. Berendzen, K. Chu, R. M. Sweet, and I. Schlichting. The role of cavities in protein dynamics: crystal structure of a photolytic intermediate of a mutant myoglobin. *Proc. Natl. Acad. Sci. U.S.A.*, 97(5):2058–2063, Feb 2000. [PubMed:10681426] [PubMed Central:PMC15753] [doi:10.1073/pnas.040459697]. 64, 66

[15] Thanh-Tung Cao, Ashwin Nanjappa, Mingcen Gao, and Tiow-Seng Tan. A gpu accelerated algorithm for 3d delaunay triangulation. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 47–54. ACM, 2014. 103, 117

[16] Thanh-Tung Cao, Herbert Edelsbrunner, and Tiow-Seng Tan. Triangulations from topologically correct digital voronoi diagrams. *Comput. Geom. Theory Appl.*, 48(7):507–519, 2015. 82

[17] 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. [PubMed:12070144] [doi:10.1074/jbc.M201373200]. 50

[18] 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. 24

[19] Xiaolin Cheng, Ivaylo Ivanov, Hailong Wang, Steven M. Sine, and J. Andrew McCammon. Molecular-dynamics simulations of elic—a prokaryotic homologue of the nicotinic acetylcholine receptor. *Biophysical Journal*, 96(11):4502 – 4513, 2009. ISSN 0006-3495. [PubMed:19486673] [PubMed Central:PMC2711484] [doi:10.1016/j.bpj.2009.03.018]. 36

[20] E. Chovancova, A. Pavelka, P. Benes, O. Strnad, J. Brezovsky, B. Kozlikova, A. Gora, V. Sustr, M. Klvana, P. Medek, L. Biedermannova, J. Sochor, and J. Damborsky. CAVER 3.0: a tool for the analysis of transport pathways in dynamic protein structures. *PLoS Comput. Biol.*, 8(10):e1002708, 2012. [PubMed:23093919] [PubMed Central:PMC3475669] [doi:10.1371/journal.pcbi.1002708]. 14, 21

[21] K. Chu, J. Vojtchovsky, B. H. McMahon, R. M. Sweet, J. Berendzen, and I. Schlichting. Structure of a ligand-binding intermediate in wild-type carbonmonoxy myoglobin. *Nature*, 403(6772):921–923, Feb 2000. [PubMed:10706294] [doi:10.1038/35002641]. 64, 80

[22] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Dewall: A fast divide and conquer delaunay triangulation algorithm in ed. *Computer-Aided Design*, 30(5):333–341, 1998. 103

[23] R. G. Coleman and K. A. Sharp. Finding and characterizing tunnels in macromolecules with application to ion channels and pores. *Biophys. J.*, 96(2):632–645, Jan 2009. [PubMed:18849407] [PubMed Central:PMC2716472] [doi:10.1529/biophysj.108.135970]. 14

[24] P. J. Corringer, F. Poitevin, M. S. Prevost, L. Sauguet, M. Delarue, and J. P. Changeux. Structure and pharmacology of pentameric receptor channels: from bacteria to brain. *Structure*, 20(6):941–956, Jun 2012. [PubMed:22681900] [doi:10.1016/j.str.2012.05.003]. 36

[25] Tran Kai Frank Da, Sébastien Loriot, and Mariette Yvinec. 3D alpha shapes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.11 edition, 2017. URL `http://doc.cgal.org/4.11/Manual/packages.html#PkgAlphaShapes3Summary`. 104

[26] C. J. Dacosta and J. E. Baenziger. Gating of pentameric ligand-gated ion channels: structural insights and ambiguities. *Structure*, 21(8):1271–1283, Aug 2013. [PubMed:23931140] [doi:10.1016/j.str.2013.06.019]. 36

[27] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000. 89

[28] W.L. DeLano. The pymol molecular graphics system. http://www.pymol.org, 2002. [PubMed:27818847] [PubMed Central:PMC5091839] [doi:10.1109/ICCABS.2011.5729867]. 25, 52

[29] 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 Res.*, 34(Web Server issue):W116–118, Jul 2006. [PubMed:16844972] [PubMed Central:PMC1538779] [doi:10.1093/nar/gkl282]. 14

[30] 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. [PubMed:16844972] [PubMed Central:PMC1538779]. 50, 66, 103

[31] H. Edelsbrunner. Biological applications of computational topology. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 1395–1412. CRC Press, 2004. 8

[32] H. Edelsbrunner. *Computational Topology. An Introduction*. Amer. Math. Soc., 2010. 2, 8, 105

[33] 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. 50

[34] H. Edelsbrunner and P. Koehl. The geometry of biomolecular solvation. *Combinatorial & Computational Geometry*, 52:243–275, 2005. 50, 103

[35] H. Edelsbrunner and E.P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, 1994. 102, 104

[36] 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. 102

[37] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002. 19

[38] Herbert Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, New York, NY, USA, 2001. ISBN 0-521-79309-2. 105

[39] Herbert Edelsbrunner. Alpha shapes – a survey. In R. van de Weygaert, G. Vegter, J. Ritzerveld, and V. Icke, editors, *Tesellations in the Sciences: Virtues, Techniques and Applications of Geometric Tilings*, 2011. 102

[40] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., USA, 2010. 11, 15

[41] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (TOG)*, 9(1): 66–104, 1990. 106

[42] Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1(1):25–44, Mar 1986. 103

[43] Herbert Edelsbrunner and Nimish R Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15(3):223–241, 1996. 103

[44] E. Eren, J. Vijayaraghavan, J. Liu, B. R. Cheneke, D. S. Touw, B. W. Lepore, M. Indic, L. Movileanu, and B. van den Berg. Substrate specificity within a family of outer membrane carboxylate channels. *PLoS Biol.*, 10(1):e1001242, Jan 2012. [PubMed:22272184] [PubMed Central:PMC3260308] [doi:10.1371/journal.pbio.1001242]. 40

[45] N. Furnham, G. L. Holliday, T. A. de Beer, J. O. Jacobsen, W. R. Pearson, and J. M. Thornton. The Catalytic Site Atlas 2.0: cataloging catalytic sites and residues identified in enzymes. *Nucleic Acids Res.*, 42(Database issue):D485–489, Jan 2014. [PubMed:24319146] [PubMed Central:PMC3964973] [doi:10.1093/nar/gkt1243]. 19, 29

[46] E. Gasteiger, C. Hoogland, A. Gattiker, S. Duvaud, M. R. Wilkins, R.D. Appel, and A. Bairoch. Protein identification and analysis tools on the expasy server. In *The Proteomics Protocols Handbook*, pages 571–607. 2005. [PubMed:10027275]. 22

[47] Leonidas J Guibas, Donald E Knuth, and Micha Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7(1):381–413, 1992. 103

[48] Dan Halperin and Mark H Overmars. Spheres, molecules, and hidden surface removal. *Computational Geometry*, 11(2):83–102, 1998. 114

[49] M. Hendlich, F. Rippmann, and G. Barnickel. LIGSITE: automatic and efficient detection of potential small molecule-binding sites in proteins. *J. Mol. Graph. Model.*, 15(6):359–363, Dec 1997. [PubMed:9704298]. 14, 50

[50] R. J. Hilf and R. Dutzler. A prokaryotic perspective on pentameric ligand-gated ion channel structure. *Curr. Opin. Struct. Biol.*, 19(4):418–424, Aug 2009. [PubMed:19646860] [doi:10.1016/j.sbi.2009.07.006]. 37

[51] B. K. Ho and F. Gruswitz. HOLLOW: generating accurate representations of channel and interior surfaces in molecular structures. *BMC Struct. Biol.*, 8:49, 2008. [PubMed:19014592] [PubMed Central:PMC2603037] [doi:10.1186/1472-6807-8-49]. 14

[52] Kenneth E Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 277–286. ACM Press/Addison-Wesley Publishing Co., 1999. 82, 85

[53] C. A. Hubner and T. J. Jentsch. Ion channel diseases. *Hum. Mol. Genet.*, 11(20):2435–2445, Oct 2002. 13

[54] M. Karpusas, W. A. Baase, M. Matsumura, and B. W. Matthews. Hydrophobic packing in T4 lysozyme probed by cavity-filling mutants. *Proc. Natl. Acad. Sci. U.S.A.*, 86(21):8237–8241, Nov 1989. [PubMed:2682639] [PubMed Central:PMC298255]. 49

[55] Deok-Soo Kim, Youngsong Cho, Jae-Kwan Kim, and Kokichi Sugihara. Tunnels and voids in molecules via voronoi diagrams and beta-complexes. *Transactions on Computational Science*, 20:92–111, 2013. 14, 21

[56] P. Koehl, M. Levitt, and H. Edelsbrunner. Proshape: understanding the shape of protein structures. *Software at biogeometry.duke.edu/software/proshape*, 2004. 14, 50

[57] B. Kozlikova, E. Sebestova, V. Sustr, J. Brezovsky, O. Strnad, L. Daniel, D. Bednar, A. Pavelka, M. Manak, M. Bezdeka, P. Benes, M. Kotry, A. Gora, J. Damborsky, and J. Sochor. CAVER Analyst 1.0: graphic tool for interactive visualization and analysis of tunnels and channels in protein structures. *Bioinformatics*, 30(18):2684–2685, Sep 2014. [PubMed:24876375] [doi:10.1093/bioinformatics/btu364]. 14, 25, 30

[58] A. Krogh, B. Larsson, G. von Heijne, and E. L. Sonnhammer. Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes. *J. Mol. Biol.*, 305(3):567–580, Jan 2001. [PubMed:11152613] [doi:10.1006/jmbi.2000.4315]. 20

[59] 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. 97

[60] M. Krone, B. Kozlikova, N. Lindow, M. Baaden, D. Baum, J. Parulek, H.-C. Hege, and I. Viola. Visual analysis of biomolecular cavities: State of the art. *Computer Graphics Forum*, 35(3):527–551, 2016. [doi:10.1111/cgf.12928]. 1, 50, 103

[61] J. Kyte and R. F. Doolittle. A simple method for displaying the hydropathic character of a protein. *J. Mol. Biol.*, 157(1):105–132, May 1982. [PubMed:7108955]. 22

[62] Cheolju Lee, Soon-Ho Park, Min-Youn Lee, and Myeong-Hee Yu. Regulation of protein function by native metastability. *Proceedings of the National Academy of Sciences*, 97 (14):7727–7731, 2000. doi: 10.1073/pnas.97.14.7727. [PubMed:10884404] [PubMed Central:PMC16612]. 49

[63] D. G. Levitt and L. J. Banaszak. POCKET: a computer graphics method for identifying and displaying protein cavities and their surrounding amino acids. *J Mol Graph*, 10(4):229–234, Dec 1992. [PubMed:1476996]. 14, 50

[64] 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. [PubMed:9741840]. 11, 103

[65] 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. [PubMed:9741841]. 11, 50, 103

[66] J. Liang, H. Edelsbrunner, and C. Woodward. Anatomy of protein pockets and cavities. *Protein Science*, 7(9):1884–1897, 1998. [PubMed:9761470] [PubMed Central:PMC2144175] [doi:10.1002/pro.5560070905]. 11, 50, 103

[67] M. Liao, E. Cao, D. Julius, and Y. Cheng. Structure of the TRPV1 ion channel determined by electron cryo-microscopy. *Nature*, 504(7478):107–112, Dec 2013. [PubMed:24305160] [PubMed Central:PMC4078027] [doi:10.1038/nature12822]. 37

[68] 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. [PubMed:22034320] [doi:10.1109/TVCG.2011.259]. 14, 20, 21, 50, 69, 81, 97

[69] 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. [doi:10.1109/BioVis.2012.6378599]. 14, 97

[70] J. Liu, A. J. Wolfe, E. Eren, J. Vijayaraghavan, M. Indic, B. van den Berg, and L. Movileanu. Cation selectivity is a conserved feature in the OccD subfamily of Pseudomonas aeruginosa. *Biochim. Biophys. Acta*, 1818(11):2908–2916, Nov 2012. [PubMed:22824298] [PubMed Central:PMC3424372] [doi:10.1016/j.bbamem.2012.07.009]. 40

[71] M. A. Lomize, I. D. Pogozheva, H. Joo, H. I. Mosberg, and A. L. Lomize. OPM database and PPM web server: resources for positioning of proteins in membranes. *Nucleic Acids Res.*, 40 (Database issue):D370–376, Jan 2012. [PubMed:21890895] [PubMed Central:PMC3245162] [doi:10.1093/nar/gkr703]. 20

[72] Paul Mach and Patrice Koehl. Geometric measures of large biomolecules: Surface, volume, and pockets. *Journal of Computational Chemistry*, 32(14):3023–3038, 2011. [PubMed:21823134] [PubMed Central:PMC3188685] [doi:10.1002/jcc.21884]. 21, 103, 104, 117

[73] Talha Bin Masood and Vijay Natarajan. An integrated geometric and topological approach to connecting cavities in biomolecules. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pages 104–111. IEEE, 2016. [doi:10.1109/PACIFICVIS.2016.7465257]. 4, 103

[74] Talha Bin Masood, Hari Krishna Malladi, and Vijay Natarajan. Facet-JFA: Faster computation of discrete voronoi diagrams. In *Proc. Indian Conference on Computer Vision Graphics and Image Processing*, ICVGIP '14, pages 20:1–20:8. ACM, 2014. ISBN 978-1-4503-3061-9. [doi:10.1145/2683483.2683503]. 6, 50

[75] Talha Bin Masood, Sankaran Sandhya, Nagasuma R. Chandra, and Vijay Natarajan. ChExVis: a tool for molecular channel extraction and visualization. *BMC Bioinformatics*, 16: 119, 2015. [PubMed:25888118] [PubMed Central:PMC4411761] [doi:10.1186/s12859-015-0545-9]. 3, 50, 97, 103

[76] Daniel L Minor. Puzzle plugged by protein pore plasticity. *Molecular cell*, 26(4):459–460, 2007. [PubMed:17531803] [doi:10.1016/j.molcel.2007.05.003]. 63, 64, 78

[77] Ashwin Nanjappa. *Delaunay triangulation in* $\mathbb{R}^3$ *on the GPU*. PhD thesis, National University of Singapore, 2012. 103

[78] J. Parulek, C. Turkay, N. Reuter, and I. Viola. Implicit surfaces for interactive graph based cavity analysis of molecular simulations. In *Biological Data Visualization (BioVis), 2012 IEEE Symposium on*, pages 115–122, 2012. [doi:10.1109/BioVis.2012.6378601]. 50, 97

[79] M. Pellegrini-Calace, T. Maiwald, and J. M. Thornton. PoreWalker: a novel tool for the identification and characterization of channels in transmembrane proteins from their three-dimensional structure. *PLoS Comput. Biol.*, 5(7):e1000440, Jul 2009. [PubMed:19609355] [PubMed Central:PMC2704872] [doi:10.1371/journal.pcbi.1000440]. 14, 20, 25

[80] 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. [PubMed:17997961] [doi:10.1016/j.str.2007.10.007]. 14, 20, 50, 69, 81, 97

[81] 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. [PubMed:16792811] [PubMed Central:PMC1539030] [doi:10.1186/1471-2105-7-316]. 50, 69, 81, 97

[82] M. Raunest and C. Kandt. dxTuber: detecting protein cavities, tunnels and clefts based on protein and solvent dynamics. *J. Mol. Graph. Model.*, 29(7):895–905, Jun 2011. [PubMed:21420887] [doi:10.1016/j.jmgm.2011.02.003]. 14

[83] Frederic M. Richards. The interpretation of protein structures: Total volume, group volume distributions and packing density. *Journal of molecular biology*, 82(1):1 – 14, 1974. ISSN 0022-2836. [PubMed:4818482][doi:10.1016/0022-2836(74)90570-1]. 49

[84] Guodong Rong and Tiow-Seng Tan. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 109–116. ACM Press, 2006. 82, 85, 86

[85] Guodong Rong and Tiow-Seng Tan. Variants of jump flooding algorithm for computing discrete Voronoi diagrams. In *Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD'07)*, pages 176–181, 2007. 86

[86] L. Sauguet, F. Poitevin, S. Murail, C. Van Renterghem, G. Moraga-Cid, L. Malherbe, A. W. Thompson, P. Koehl, P. J. Corringer, M. Baaden, and M. Delarue. Structural basis for ion permeation mechanism in pentameric ligand-gated ion channels. *EMBO J.*, 32(5):728–741, Mar 2013. [PubMed:23403925] [PubMed Central:PMC3590989] [doi:10.1038/emboj.2013.17]. 36

[87] D. Sehnal, R. S. Vařeková, K. Berka, L. Pravda, V. Navratilová, P. Banáš, C. M. Ionescu, M. Otyepka, and J. Koča. MOLE 2.0: advanced approach for analysis of biomacromolec-

ular channels. *J Cheminform*, 5(1):39, Aug 2013. [PubMed:23953065] [PubMed Central:PMC3765717] [doi:10.1186/1758-2946-5-39]. 13, 14, 19, 25

[88] O. S. Smart, J. G. Neduvelil, X. Wang, B. A. Wallace, and M. S. Sansom. HOLE: a program for the analysis of the pore dimensions of ion channel structural models. *J Mol Graph*, 14(6): 354–360, Dec 1996. [PubMed:9195488]. 14

[89] Raghavendra Sridharamurthy, Harish Doraiswamy, Siddharth Patel, Raghavan Varadarajan, and Vijay Natarajan. Extraction of Robust Voids and Pockets in Proteins. In Mario Hlawitschka and Tino Weinkauf, editors, *EuroVis - Short Papers*. The Eurographics Association, 2013. [doi:10.2312/PE.EuroVisShort.EuroVisShort2013.067-071]. 51, 52, 55, 75, 76

[90] Raghavendra Sridharamurthy, Talha Bin Masood, Harish Doraiswamy, Siddharth Patel, Raghavan Varadarajan, and Vijay Natarajan. Extraction of robust voids and pockets in proteins. In Lars Linsen, Hans-Christian Hege, and Bernd Hamann, editors, *Visualization in Medicine and Life Sciences III: Towards Making an Impact*, pages 329–349. Springer International Publishing, Mathematics and Visualization Series, 2016. ISBN 978-3-319-24523-2. [doi:10.1007/978-3-319-24523-2_15]. 4, 52, 55, 103

[91] K. J. Swartz. Towards a structural view of gating in potassium channels. *Nat. Rev. Neurosci.*, 5(12):905–916, Dec 2004. [PubMed:15550946] [doi:10.1038/nrn1559]. 40

[92] K. P. Tan, R. Varadarajan, and M. S. Madhusudhan. DEPTH: a web server to compute depth and predict small-molecule binding cavities in proteins. *Nucleic Acids Res.*, 39(Web Server issue):W242–248, Jul 2011. [PubMed:21576233] [PubMed Central:PMC3125764] [doi:10.1093/nar/gkr356]. 53

[93] K. P. Tan, T. B. Nguyen, S. Patel, R. Varadarajan, and M. S. Madhusudhan. Depth: a web server to compute depth, cavity sizes, detect potential small-molecule ligand-binding cavities and predict the pKa of ionizable residues in proteins. *Nucleic Acids Res.*, 41(Web Server issue):W314–321, Jul 2013. [PubMed:23766289] [PubMed Central:PMC3692129] [doi:10.1093/nar/gkt503]. 53

[94] 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. [PubMed:19626353] [doi:10.1007/s00894-009-0541-y]. 50

[95] R. F. Tilton, I. D. Kuntz, and G. A. Petsko. Cavities in proteins: structure of a metmyoglobin-xenon complex solved to 1.9 A. *Biochemistry*, 23(13):2849–2857, Jun 1984. [PubMed:6466620]. 64, 80

[96] A. Tripathi and G. E. Kellogg. A novel and efficient tool for locating and characterizing protein cavities and binding sites. *Proteins*, 78(4):825–842, Mar 2010. [PubMed:19847777] [PubMed Central:PMC2811767] [doi:10.1002/prot.22608]. 14

[97] Bert van den Berg, William M Clemons, Ian Collinson, Yorgo Modis, Enno Hartmann, Stephen C Harrison, and Tom A Rapoport. X-ray structure of a protein-conducting channel. *Nature*, 427(6969):36–44, 2004. [PubMed:14661030] [doi:10.1038/nature02218]. 63, 64, 78

[98] N. R. Voss and M. Gerstein. 3V: cavity, channel and cleft volume calculator and extractor. *Nucleic Acids Res.*, 38(Web Server issue):W555–562, Jul 2010. [PubMed:20478824] [PubMed Central:PMC2896178] [doi:10.1093/nar/gkq395]. 14

[99] David F Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The computer journal*, 24(2):167–172, 1981. 103

[100] E. Yaffe, D. Fishelovitch, H. J. Wolfson, D. Halperin, and R. Nussinov. MolAxis: efficient and accurate identification of channels in macromolecules. *Proteins*, 73(1):72–86, Oct 2008. [PubMed:18393395] [PubMed Central:PMC2693897] [doi:10.1002/prot.22052]. 14, 25, 97

[101] H. X. Zhou and J. A. McCammon. The gates of ion channels and enzymes. *Trends Biochem. Sci.*, 35(3):179–185, Mar 2010. [PubMed:19926290] [PubMed Central:PMC2867094] [doi:10.1016/j.tibs.2009.10.007]. 13

[102] M. Zhou and R. MacKinnon. A mutant KcsA K(+) channel with altered conduction properties and selectivity filter ion distribution. *J. Mol. Biol.*, 338(4):839–846, May 2004. [PubMed:15099749] [doi:10.1016/j.jmb.2004.03.020]. 41