UNIVERSITY OF CALIFORNIA
Santa Barbara

Efficient and Automated Analysis of Protein Structures

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy
in
Computer Science

by

Tolga Can

Committee in Charge:

Professor Yuan-Fang Wang, Chair
Professor Ambuj K. Singh
Professor Jianwen Su
Professor Matthew Turk

September 2004

The dissertation of Tolga Can is approved.

Professor Ambuj K. Singh

Professor Jianwen Su

Professor Matthew Turk

Professor Yuan-Fang Wang, Committee Chair

August 2004

Efficient and Automated Analysis of

Protein Structures

To my wife, Aysu Betin-Can.

Acknowledgements

I will remember my PhD career as a gratifying and engaging experience thanks to many people including the faculty and the staff in my department, my friends, my parents, and my wife.

First of all, I would like to thank Professor Yuan-Fang Wang for being a great advisor throughout my graduate study. His enthusiasm, great ideas and directions helped me stay on the right track all the time. His trust in me encouraged me to overcome the hardest problems. I was inspired by his extensive knowledge and wisdom and admired his humility. I am proud of being one of his students.

I would also like to thank Professor Ambuj K. Singh, Professor Jianwen Su, and Professor Matthew Turk for serving on my committee and for keeping their doors open all the time for any of the questions I had.

I would like to thank the Computer Science staff for all their efforts to help us achieve our goals. They made us feel as part of the family.

I really enjoyed working in Vision Lab. I would like to thank all my friends in the lab, Yujun Wang, Dan E. Koppel, Long Jiao, Wei Niu, and Xiang Fu for the fruitful discussions we had. I would also like to thank members of the DBL lab I collaborated with, Arnab Bhattacharya, Orhan Çamoğlu, and Tamer Kahveci.

I would like to thank my parents and my sister for supporting me and believing in me all through my life as a student, even if the support had to come through once a week phone calls from overseas for the last four years. Last but not the least, I would like to thank my wife Aysu for always being with me with her endless love and support.

Curriculum Vitæ

Tolga Can

Education

September 2004	Doctor of Philosophy in Computer Science, University of California, Santa Barbara.
December 2003	Master of Science in Computer Science, University of California, Santa Barbara.
June 1998	Bachelor of Science in Computer Engineering, Middle East Technical University, Ankara, Turkey.

Fields of Study

Comparative proteomics, computer graphics, machine learning, pattern recognition, volume visualization, bio-molecular image databases, level set methods.

Publications

T. Can, O. Çamoğlu, A.K. Singh, and Y.-F. Wang
Automated Protein Classification Using Consensus Decision.
to appear in Journal of Bioinformatics and Computational Biology (JBCB), 2005.

T. Can and Y.-F. Wang
Molecular Surface Generation and Interior Cavity Detection Using Level Set Methods.
submitted to Pacific Symposium on Biocomputing (PSB), 2005.

T. Can, O. Çamoğlu, A.K. Singh, and Y.-F. Wang
Automated Protein Classification Using Consensus Decision.
Computational Systems Bioinformatics (CSB), 2004, Stanford, CA.

T. Can and Y.-F. Wang
Protein Structure Alignment and Fast Similarity Search Using Local Shape Signatures.
Journal of Bioinformatics and Computational Biology (JBCB), 2:1, pages 215–239, 2004.

A. Bhattacharya, T. Can, T. Kahveci, A. K. Singh, and Y.-F. Wang
ProGreSS: Simultaneous Searching of Protein Databases by Sequence and Structure.
Pacific Symposium on Biocomputing (PSB), 2004, Hawaii.

Y. Wang, T. Can, Y.-F. Wang, and J. Su
Personalized Annotation and Information Sharing in Protein Science with Information-Slips.
International Conference on Information and Knowledge Sharing (IKS), 2003, Scottsdale, AZ.

T. Can and Y.-F. Wang
CTSS: A Robust and Efficient Method for Protein Structure Alignment Based on Local Geometrical and Biological Features.
Computational Systems Bioinformatics (CSB), 2003, Stanford, CA.

T. Can, Y. Wang, Y.-F. Wang and J. Su
FPV: Fast Protein Visualization Using Java 3D.
Bioinformatics, 19:8, pages 913–922, 2003.

T. Can, Y. Wang, Y.-F. Wang and J. Su
FPV: Fast Protein Visualization Using Java 3D.
ACM Symposium on Applied Computing (SAC), 2003, Melbourne, FL.

T. Can, Y. Wang, Y.-F. Wang and J. Su
A Distributed Protein Visualization Application.
Georgia Tech-Emory International Conference on Bioinfomatics, (poster), 2001, Atlanta, GA.

T. Can, V. Isler and Z. Ipekkan
Sensor Optimization In a Virtual Environment.
Conference on Computer Generated Forces and Behavioral Representation, 2000, Orlando, FL.

Abstract

Efficient and Automated Analysis of Protein Structures

by

Tolga Can

In recent years, computational complexity in structural bioinformatics attained a new level with the vast increase in the amount of structural data available. The Protein Data Bank (PDB), which is the single worldwide repository for 3-D macromolecular structure data, contains more than 25k structures as of July 2004. However, existing methods for protein structure analysis are unable to cope with this increase in the amount of available data. Therefore, this wealth of data requires computationally efficient methods to be developed for the analysis of large numbers of protein structures and their associated functions.

In this dissertation, we present methods for protein structure analysis that can scale well with the amount of protein structure data available. Our work can be described under three main categories: (1) visualization and surface modelling, (2) structure comparison and similarity search, and (3) automated classification.

For efficiently visualizing protein structures using a scene-graph based graphics API, we have developed methods to optimize the constructed scene-graph to enable real-time visualization of very large protein complexes. Our method (FPV) achieves up to 8 times interactive speed compared to existing methods. For generation of molecular surfaces we recently developed a method based on a level set formulation that can compute the surface and interior inaccessible cavities very efficiently (1.5 to 3.14 times faster on the average than compared methods).

For comparison and similarity search of protein structures we have developed a method that utilizes local shape signatures based on the theory of differential geometry. Our method (CTSS) is up to 30 times faster than CE, a widely used method for structure comparison, while achieving the similar level of accuracy. We have also developed an integrated sequence and structure analysis method (ProGreSS), which enables biologists to perform joint sequence and structure similarity queries while improving on the accuracy and efficiency of existing methods.

For an up-to-date view of the protein structure universe with the help of automated classification, we have developed an ensemble classifier based on decision

trees rooted in machine learning. We show that higher classification accuracy can be achieved using the joint hypothesis of the ensemble classifier.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

After the completion of the Human Genome Project [14], considerable effort is being expended on structural genomics research with the aim of determining the structure and function of as many gene products possible. The genomic data, combined with the considerable amount of current structural data (26485 protein structures as of July 27, 2004 at the Protein Data Bank [8]), will lead to the emergence of protein structure analysis as a critical component in finding answers to some long-standing, fundamental questions on how cells function, what the action pathways of certain disease agents are, and what make human beings who we are.

Existing methods for protein structure analysis are developed usually without consideration of the increasing amount of available data. As a result of this, many of the methods fail to give satisfactory performance when efficiency and interactivity is of concern. To give an example, most of the widely used structure similarity methods cannot perform an online query, instead they report results that are computed off-line or report the results through e-mail. It is clear that this wealth of data requires computationally efficient methods to be developed for the analysis of large numbers

of protein structures and their associated functions.

In this introductory chapter, we give the biological background necessary for computer scientists to understand the material presented in the proceeding chapters. We also give the overall picture of the area of computational protein structure analysis and briefly describe the specific problems of protein structure comparison, surface modeling, and protein classification.

1.1 Biological Background

In this section we first give an overview of the source of information in computational biology. Most of the problems in computational biology focus on three primary sources of data: DNA or protein sequences, macromolecular structures and the results of functional genomic experiments. Raw DNA sequences are strings of the four base-letters (A, G, C, T) comprising genes, each typically 1,000 bases long. The GenBank [7] repository for nucleic acid sequences hold approximately 28,507,990,166 bases in 22,318,883 sequence records as of January 2003. At the next level are the protein sequences comprising strings of 20 amino-acid letters. At present there are about 938,390 known protein sequences [12] as of September 2003, with a typical bacterial protein containing approximately 300 amino acids. Macromolecular structural data represents a more complex form of information. There are currently 26,485 entries in the Protein Data Bank (PDB) [8] as of July 2004, containing atomic structures of proteins, DNA and RNA solved by x-ray crystallography and NMR methods. A typical PDB file for a medium-sized protein contains

the three-dimensional coordinates of approximately 2,000 atoms.

Analyzing new protein structures published in the Protein Data Bank may reveal many unexpected functional and evolution relationships that were hidden at the sequence level. Also, the computation of a molecular surface is essential in determining the surface residues of a protein that is in contact with the outer environment of the protein structure. In Chapters 3 and 4, we present methods for interactively analyzing the three-dimensional protein structure and computing its solvent-accessible and solvent-excluded surface.

An essential aspect of managing this large volume of biological data lies in developing methods for assessing similarities between different biomolecules and identifying those that are related. The algorithms we present in Chapters 5, 6, and 7 of this dissertation are mainly of this category: analyzing the structure data and inferring relationship to a large database of structures either by finding a list of similar structures or classifying the input structure into a category of a hierarchical classification of the whole database.

1.1.1 Introduction to Protein Structures

Proteins are not linear molecules as suggested when the protein sequence is described as a *string* of amino acid letters, -Lys-Ala-Pro-Met-Gly- etc., for example. Rather, this *string* folds into an intricate three-dimensional structure that is unique to each protein. It is this three-dimensional structure that allows proteins to function. Thus, in order to understand the details of protein function, one must understand protein structure.

In order to fully explore protein structure in detail a number of different types of molecular models are used. Figure 1.1 shows four different commonly used models for the same protein molecule, 1d9c. The details of these models are discussed in the later chapters before the details of the methods that generate these models are presented.

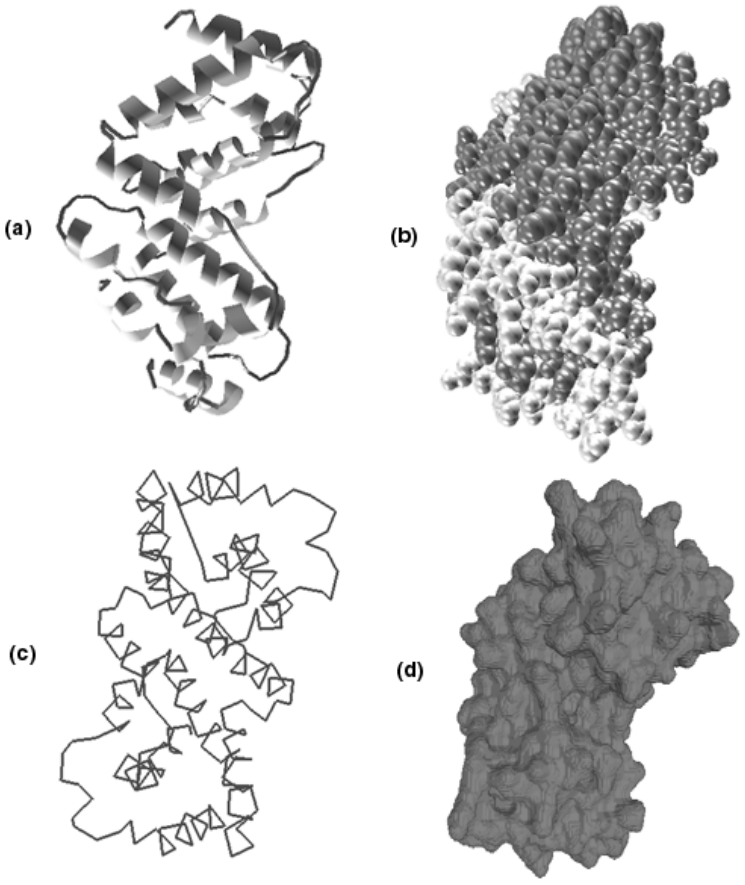

Figure 1.1: The *ribbon* model (a), the *spacefill* model (b), the C_α trace (c), and the *solvent-excluded surface* (d).

Protein structure is broken down into four levels:

- **Primary structure:** refers to the *linear* sequence of amino acids. Proteins are large polypeptides of defined amino acid sequence. The sequence of amino acids in each protein is determined by the gene that encodes it. The gene is transcribed into a messenger RNA (mRNA) and the mRNA is translated into a protein by the ribosome. Primary structure is sometimes called the *covalent structure* of proteins because, with the exception of disulfide bonds, all of the covalent bonding within proteins defines the primary structure. In contrast, the higher orders of proteins structure (i.e. secondary, tertiary and quaternary) involve mainly noncovalent interactions.

- **Secondary structure:** is *local* ordered structure brought about via hydrogen bonding mainly within the peptide backbone. The most common secondary structure elements in proteins are the alpha helix and the beta sheet (sometimes called beta pleated sheet).

- **Tertiary structure:** is the *global* folding of a single polypeptide chain. A major driving force in determining the tertiary structure of globular proteins is the hydrophobic effect. The polypeptide chain folds such that the side chains of the nonpolar amino acids are *hidden* within the structure and the side chains of the polar residues are exposed on the outer surface. Hydrogen bonding involving groups from both the peptide backbone and the side chains are important in stabilizing tertiary structure. The tertiary structure of some proteins is stabilized by disulfide bonds between cysteine residues.

- **Quaternary structure:** involves the association of two or more polypeptide chains into a multi-subunit structure. Quaternary structure is the stable association of multiple polypeptide chains resulting in an active unit. Not all proteins exhibit quaternary structure. Usually, each polypeptide within a multi-subunit protein folds more-or-less independently into a stable tertiary structure and the folded subunits then associate with each other to form the final structure.

1.2 Overview of Dissertation

In Chapter 2, we give the details of the scalability problems related to protein structure analysis. The methodologies we have developed to solve these scalability problems are presented in three parts. The first part, consisting of Chapters 3 and 4, deals with computation of different structural representations of protein molecules. In Chapter 3, we present methods for efficiently visualizing protein structures using a scene-graph based graphics API. In Chapter 4 we propose an efficient level-set based method for computing the solvent-accessible surface and interior inaccessible cavities of a protein structure. The second part, Chapters 5 and 6, considers the problem of similarity searches in large protein databases. In Chapter 5 we explain in detail the methods we have developed for extracting structural features of protein molecules and aligning them by using geometric hashing techniques. In Chapter 6 we show how joint sequence and structural alignment can be performed by using a novel index structure. The third part, Chapter 7, presents an ensemble classifier framework for automated classification of protein structures. Finally, in Chapter 8,

we conclude with a brief summary of the dissertation and future directions.

Chapter 2

Scalability Issues in Protein Structure Analysis

Understanding a protein's structure with the help of protein structure analysis tools helps researchers to better determine how the protein works in its biological role. Analysis of a protein structure also involves studying the interaction of that protein with other protein structures, because proteins do not perform their functions in isolation. Furthermore, a global view of the protein structure universe is important to have a better understanding of this huge protein interaction network and also of the relationships of proteins from different species.

The Protein Data Bank (PDB) [8] is the single worldwide repository for protein structure data, and contains more than 25k structures as of July 2004. Currently, about 100 new protein structures per week are published in the PDB (226 new structures between July 27, 2004 and August 10, 2004 updates). Furthermore, with the advances in experimental structure determination technologies, the structural data of large protein/DNA, protein/RNA, or protein/protein complexes can be determined

accurately and made available at the Protein Data Bank. However, existing methods for protein structure analysis are unable to cope with both the increase in the amount of available data and the increase in the size of structural data to be analyzed.

In this chapter we discuss the scalability problems encountered in protein structure analysis related to both the growth in protein structure databases and the increase in size of analyzed data.

2.1 Growth of Protein Structure Databases

Despite the tremendous cost and time required for experimental determination of protein structures, protein structure databases grow at a considerably high rate. It should be also noted that the ratio of protein structures with known macromolecular structure to the total repertoire of protein sequences is very low: currently about 5%. This ratio also tells us that, if we knew the structure of every protein sequence as produced by the genome sequencing projects, the structure databases would be about 20 times larger than current size of the Protein Data Bank (as of July 2004). This wealth of data raises significant questions about the ability of existing methods in handling structure similarity search and classification in a reasonable amount of time.

Most of the existing methods [80, 42, 60] for protein structure comparison are designed for pairwise comparison. Therefore, in order to conduct a similarity search of a protein structure against a database of protein structures, one needs to perform an exhaustive search by pairwise comparing the query protein to all of the database

proteins one by one. For example, if we use CE [80] for such a similarity query against the current Protein Data Bank, this would take about 7 hours on an Intel Pentium 4 Processor at 2.0GHz and 512MB of RAM, where a pairwise comparison takes about 1 second. Furthermore, if one wants to perform an all-to-all comparison of the Protein Data Bank for obtaining a global picture of the relationships among proteins, this would take about 10 years on that same single processor machine. Clearly, this is not a reasonable approach, and methods for conducting fast similarity searches should be developed.

A global view of the protein structure universe is also established with the help of structural classification databases [44, 68, 64]. These databases are used to define the relationships—in terms of sequence, structure, and function—of proteins. Of these classification schemes, SCOP [64] is created mainly by manual inspection. This is perhaps the reason that it is accepted by many researchers as the most accurate classification scheme (or the ground truth). However, SCOP is updated every six months since it is quite a labor intensive process to manually place a protein structure into the correct category in a hierarchical classification of 25K protein structures. Furthermore, a 100 protein per week growth rate means about 2600 protein structures in six months. Therefore, if one requires a dynamic, up-to-date view of the protein structure universe, accurate automated classification techniques should be developed to aid in manual classification process.

2.2 Increase in Size of the Analyzed Data

One of the long-term goals of the NIH Roadmaps Structural Biology initiative is to provide structural information on large, macromolecular complexes[1]. This goal does not seem very far with the advances in experimental structure determination technologies. As a result growing number of the structural data of large protein/DNA, protein/RNA, or protein/protein complexes will be determined accurately and made available at the Protein Data Bank.

In order to understand how protein machines, i.e., complexes, work—and to figure out how to fix them when they do not—researchers need to view the protein complexes in several different orientations and using several different models, mimicking the way these assemblies twist and bend inside living cells. However, current visualization methods are not able to cope with the growing size of data to be analyzed. Large complexes such as virus capsids, ribosomes, and chromosomes can contain as many as 40K–100K atoms in their structure data. The increasing complexity of the protein structure model causes problems both in building a three-dimensional model for the molecular complex and in interacting with the rendered three-dimensional model. We show in the proceeding chapters that the space (memory) requirement of some of the methods are so large that they cannot even build a three-dimensional model of some large molecular complexes. Furthermore, even if they can build a three-dimensional model, the interactive rendering performance drops drastically, i.e., less than 1 frames per second, as the size of the molecular complexes get larger. Therefore, development of efficient methods that scales

[1] `http://nihroadmap.nih.gov/structuralbiology/index.asp`

well with the size of molecular complexes is crucial. One of the existing methods, Chimera [45], provides an extension, *the Multiscale Extension*, to its base protein analysis framework to cope with increasing size of molecular complexes. However, as we demonstrate in the proceeding chapters, new methods should be developed that can scale well with the size of molecular complexes.

Chapter 3

Efficient Visualization of Large Molecular Complexes

In this chapter we present the methods we have developed to optimize scene-graphs for efficiently visualizing large protein structures [16]. The protein visualization system presented here is based on Java 3D™. Java 3D provides compatibility among different systems and enables applications to be run remotely through web browsers. However, using Java 3D for visualization has some performance issues with it. The primary concerns about molecular visualization tools based on Java 3D are in their being slow in terms of interaction speed and in their inability to load large molecular complexes. This behavior is especially apparent when the number of atoms to be displayed is huge, or when several proteins are to be displayed simultaneously for comparison. Large complexes such as virus capsids, ribosomes, and chromosomes can contain as many as 40K–100K atoms in their structure data. This increasing complexity of the protein structure model causes problems both in building a three-dimensional model for the molecular complex and in interacting with the rendered

three-dimensional model. In this chapter we present techniques for organizing a Java 3D scene graph to tackle these problems. We demonstrate the effectiveness of these techniques by comparing the visualization component of our system with two other Java 3D based molecular visualization tools. In particular, for van der Waals display mode, with the efficient organization of the scene graph, we could achieve up to eight times improvement in rendering speed and could load molecules three times as large as the previous systems could.

3.1 Introduction

Protein visualization has become an important research topic, especially in light of the accomplishment of the Human Genome Project [14]. The ability to visualize the 3D structure of proteins is critical in many areas such as drug design and protein modeling. This is because the 3D structure of protein determines its interaction with other molecules, hence its function, and the relation of the protein to other known proteins. For example, hemoglobin's cup shape, which accommodates the oxygen-binding heme group, suggests its ability to carry oxygen in the bloodstream. There are many well established ways of visualizing the 3D protein structures. Each way of visualization highlights a different aspect of the protein molecule, as mentioned by Clay Shirky [82].

Growing number of new structure data in Protein Data Bank open new ways for collaboration, thus emphasizes the need for visualization tools that are portable. Moreover, studying the interaction between protein molecules may also require vi-

sualizing huge numbers of atoms, thus researchers also need tools that are capable of loading and displaying this huge amount of data.

3.2 Related Work

Many tools have been developed to visualize a protein whose structure has been determined. In this section we will talk about a subset of these tools, which are closely related to our molecular visualization system. One of the earliest of those tools is Roger Sayle's RasMol [77]. RasMol is now being developed under the name of Protein Explorer. Swiss-PdbViewer [37], which is tightly linked to the automated protein modeling server Swiss-Model, provides a user-friendly interface to analyze several proteins at the same time. MOLMOL [50] is another molecular graphics program for the display, analysis, and manipulation of the 3D structures of biological macromolecules, with special emphasis on nuclear magnetic resonance (NMR) solution structures of proteins and nucleic acids. Most of these programs are implemented using C language and OpenGL API and they have relatively large user communities.

There are relatively few protein visualization tools which were developed using Java and the Java 3D API. WebMol [88] is a protein structure viewing and analysis program, which has more functionality, but limited 3D model types. These two programs do not use the Java 3D API; instead they use their own graphics constructs based on Java.

JIMD Interactive Molecular Dynamics with Java[1], is being developed using Java 3D, but their focus is on molecular dynamics and simulation. Tripos Java3D Molecule Viewer[2], is a new tool currently under development. JMV and JMVS2 (two systems we used for performance comparison) are molecular visualization tools and offer a variety of 3D representations and display options. JMV is developed by the Theoretical Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign with NIH support. These two tools have very similar functionality compared to our molecular visualization system. One advantage of JMV over JMVS2 is that it is being build as a toolkit, so that other developers can use it as part of their systems.

Molecular Biology Toolkit[3] is another general toolkit that includes visualization components based on Java3D. However, it is still an ongoing work and right now no visualization application using this toolkit is available for evaluation and testing purposes.

3.3 Why do we need a new visualization tool?

Many tools have been developed to visualize protein structures. Tools that have been based on Java 3D™ are compatible among different systems and they can be run remotely through web browsers. However, using Java 3D for visualization has some performance issues with it. The primary concerns about molecular visualization

[1] `http://www.gwdg.de/~ovormoo/jimd/`
[2] ftp://ftp.tripos.com/pub/java3d
[3] `http://mbt.sdsc.edu/`

tools based on Java 3D are in their being slow in terms of interaction speed and in their inability to load large molecules. This behavior is especially apparent when the number of atoms to be displayed is huge, or when several proteins are to be displayed simultaneously for comparison.

There is growing trend in adopting the Java™ technology in the fields of bioinformatics and computational biology [62]. The main advantages of Java are its compatibility across different systems/platforms and having the ability to be run remotely through web browsers. Using Java 3D as a graphics engine has also the additional advantage of rapid application development, because Java 3D API incorporates a high-level scene graph model that allows developers to focus on the objects and the scene composition. Java 3D also promises high performance, because it is capable of taking advantage of the graphics hardware in a system. The speed observed should depend on the quality of the graphics hardware on the machine. However, a common complaint about visualization systems based on Java 3D is their being slow in terms of interaction speed even with a good graphics hardware accelerator. Also memory errors may be seen even with a small number of objects. The reason for these anomalies may be the developer himself (constructing a bad scene graph) or certain limitations of the Java 3D API, which is discussed below.

The Java 3D API implementations are layered on top of the existing lower-level immediate-mode [28] 3D rendering APIs, such as OpenGL and Direct3D. Java 3D is fundamentally a scene-graph-based API. Most of the constructs in the API are biased toward retained mode and compiled-retained mode rendering [85]. Java 3D itself also offers immediate-mode rendering if a developer wants more control and

flexibility. The programmer can ignore the scene graph structure and send the graphical constructs directly to the renderer. However, in immediate mode, Java 3D has no high-level information concerning graphical objects or their composition. Because it has minimal global knowledge, Java 3D can only perform localized optimizations on behalf of the programmer. Thus, using immediate-mode directly may cause drastic performance drops. Using a scene-graph-based development scheme a developer should expect better performance, but some molecular scenes (e.g. containing too many atoms) may require too much memory or computation time. Thus, performance drops occur because of an heavyweight scene graph. In this section we explain the techniques to create efficient scene graph structures, which allow loading large molecules (more than 4000 amino acids) and render them in an acceptable interactive speed. We demonstrate the effectiveness of the developed techniques by comparing the visualization component of our system with two other Java 3D based molecular visualization tools. In particular, for van der Waals display mode, with the efficient organization of the scene graph, we could achieve up to eight times improvement in rendering speed and could load molecules three times as large as the previous systems could.

3.4 Protein Visualization

In this section, we briefly discuss how we create molecular scenes from the protein data. We also present two accompanying textual views, which are helpful in browsing the amino acid sequence and viewing the hierarchical organization of the protein

data. The techniques for expediting rendering based on Java 3D will be discussed in Section 3.5.

3.4.1 Data

PDB files are obtained from the Protein Data Bank (PDB) [8], which is an archive of experimentally determined 3D structures of biological macromolecules. PDB files contain 3D coordinates of each atom of the protein molecule. We use these 3D coordinates and atom types to calculate the bonding information and to estimate the secondary structure. This information is needed for some of the 3D molecular representations described below.

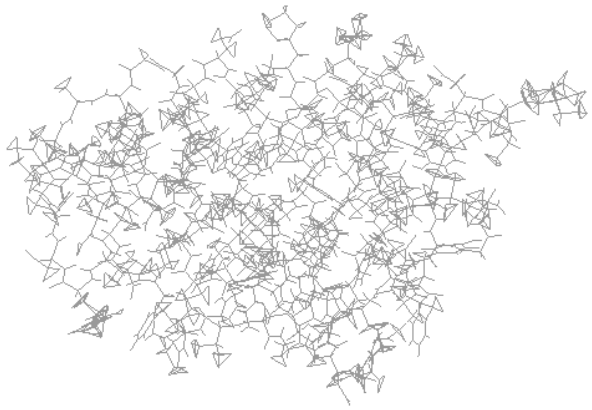

Figure 3.1: The *bonds* model.

3.4.2 3D Representations

Each representation of a protein molecule highlights a different aspect of the structure. They have advantages and disadvantages compared to each other. For example, the space-fill model can be helpful in understanding the volume a protein molecule occupies, but it lacks information about how amino acids are connected to each other, i.e. how the chain is formed. We describe below different 3D models provided by our visualization system, and explain their use and the way they are built.

Bonds Model: Bonds model is created as a wire-frame model representing the bonding information in the protein molecule. Figure 3.1 shows a *bonds* representation of the molecule Oxygen Binding (PDB ID: 2mhr).

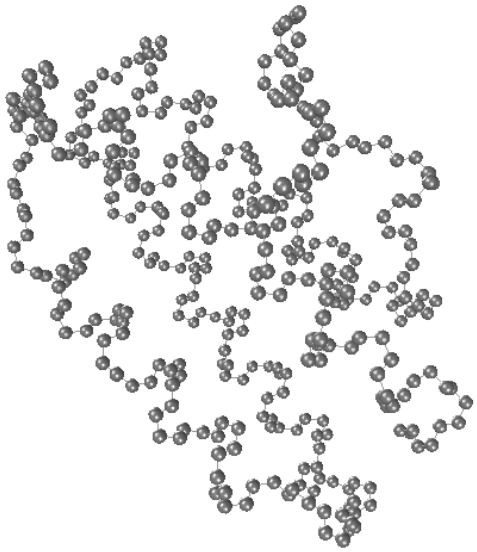

Figure 3.2: The *backbone* model.

Backbone Model: The backbone model is created by using the *alpha carbon, carbon,* and *nitrogen* atoms in the molecule. The position of the atoms are used to transform the spheres that represent them. The backbone bonds within each amino acid and the peptide bonds (between amino acids) are also shown in the model. This model is useful for understanding the protein molecule as a chain, and realizing amino acids' positions in this chain.

Figure 3.2 shows the backbone model of the molecule Oxygen Binding (PDB ID: 2mhr). When we interact with the 3D model of the backbone of a molecule, we can easily recognize how the amino acid sequence is formed in four parallel helices.

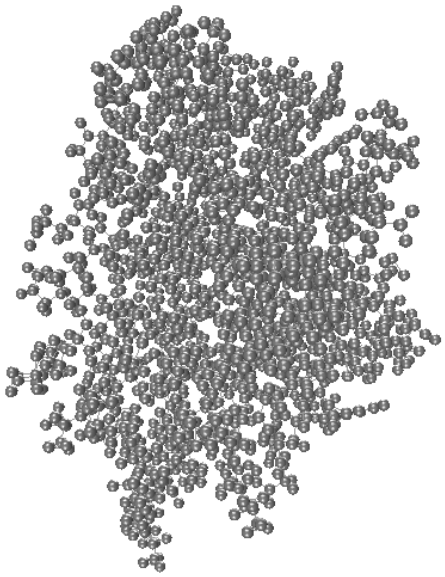

Figure 3.3: The *balls and sticks* model.

Balls and Sticks Model: The balls-sticks model shown in Figure 3.3 represents

all of the existing bonds in the molecule as sticks and all the atoms as equal sized spheres.

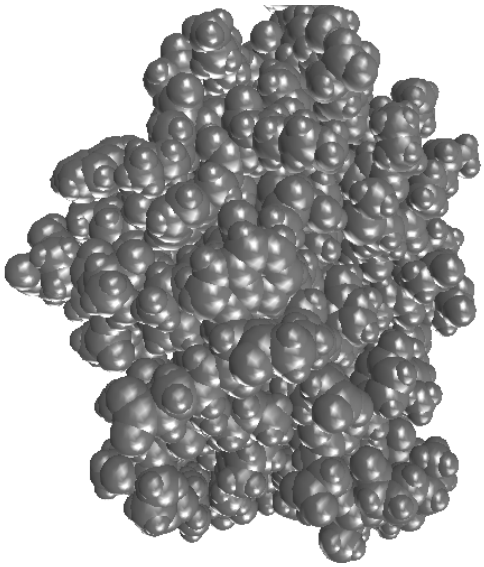

Figure 3.4: The *spacefill* model.

Space-fill (van der Waals) Model: The space-fill model is useful in visualizing the volume a protein molecule occupies (see Figure 3.4. It gives an overall view of the molecule and thus provides a good view of the tertiary structure. In this model each atom is modeled using its van der Waals radius, so that the viewer gets an idea of the relative sizes of the atoms making up the protein molecule. The atoms are represented by concrete spheres centered at the corresponding atomic coordinates read from the PDB file.

Ribbon Model: The ribbon model is used to display the secondary structures in the protein molecule. The secondary structure is predicted from the atomic coordinates

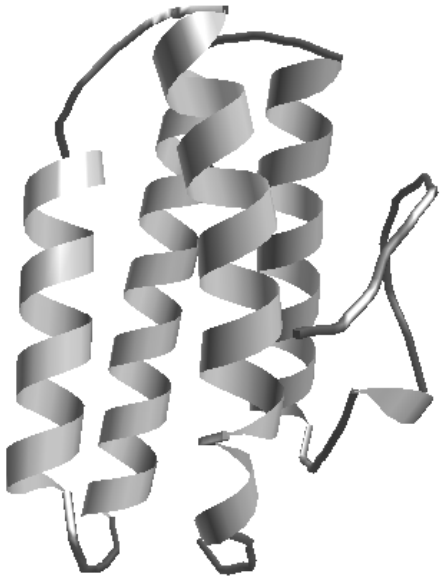

Figure 3.5: The *ribbon* model.

in the PDB file, by using the algorithm developed by Kabsch and Sander [48]. The ribbon model is created using hermite curves. Our implementation is based on the program called MolScript [51]. Figure 3.5 shows the ribbon model of the same molecule 2mhr. Here, different colors for different secondary structures are used.

3.4.3 Textual Information Windows

Having a textual representation of the protein molecule has many benefits. First of all it shows the linearity of the protein structure. The name of amino acids forming the chain is provided in a sequence view. Furthermore, the underlying hierarchy of the molecule can be captured when a tree view is used. We describe below the two

accompanying information windows provided by our visualization component.

Molecule Information Window: The molecule information window contains information about molecule's name, number of amino acids it contains, the amino acid chain, the secondary structure information, and information about currently selected sub-structure. The amino acid chain is displayed using one-letter representations of the amino acids. The molecule name info is read from the PDB file. Although it is possible to gather secondary structure information also from the PDB file, because of the fact that most of the PDB files available do not contain that information, the secondary structure information is calculated by using the prediction algorithm developed by Kabsch and Sander [48]. The information about the secondary structure is also displayed using one letter codes aligned with the amino acid codes (H:helix, B:residue in isolated beta bridge, E:extended beta strand, G:310 helix, I:pi helix, T:hydrogen bonded turn, S:bend).

When the user makes selections on the molecule during the interaction with a 3D model, the corresponding part of the amino acid chain in the information window is highlighted. If the selection is in the level of atoms, the selected atom information is also displayed in the *information window*.

Figure 3.6 shows the molecule information window during interaction with the Antitumor Protein (PDB ID: 1D8V) protein. The currently selected amino acid is *Threonine*, whose one letter code is *T*, and it is the 10th amino acid in the first (and only) chain of the protein molecule. We see in the secondary structure information that this amino acid is part of a *coil*, and currently selected atom is *alpha carbon*.

Tree View Window: Although a protein is a linear structure of amino acids, there's

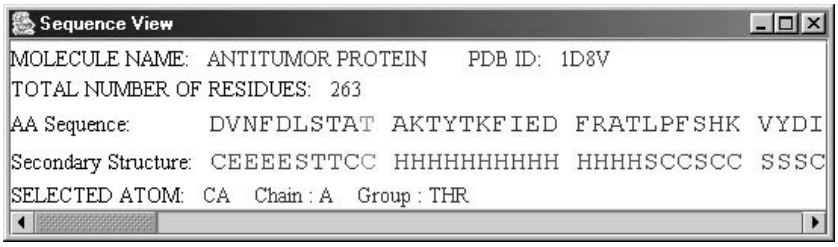

Figure 3.6: Molecule information window.

a hierarchy in the primary structure of protein molecules. A protein *molecule* is composed of one or more *chains* of amino acids. A chain may contain several *amino acids*, probably in the order of hundreds. Each amino acid has an eight atom *body* and a *side chain*, i.e. residue, which may be made up of 1 to 18 atoms. We provide a *tree view* window that visualizes this hierarchical structure of a protein molecule.

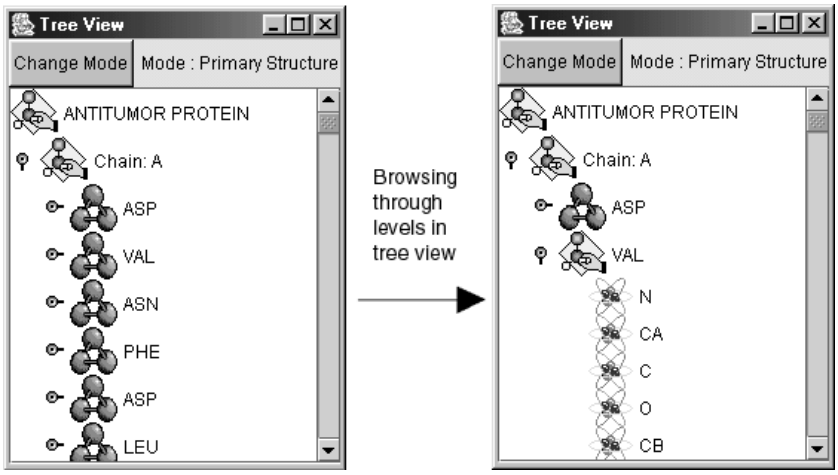

Figure 3.7: Tree-view window.

Figure 3.7 shows the tree view window while browsing through the hierarchy. In this snapshot the molecule has a very simple hierarchy, since it contains only one chain. But it is still useful to understand how the protein molecule is built. We provided a two-way interaction between the tree view and the 3D view. The user can interact with the tree by selecting its nodes. The corresponding sub-structure is highlighted in the 3D model. When the interaction is with the 3D model, and if a selection is made on it, the corresponding tree node is highlighted accordingly.

3.5 Scene-graph optimization

In this section we describe the techniques we have used to speed up real time interaction and to be able to load very large molecules. The key issue here is the way the scene graph structure is created from a protein structure file (PDB). A scene graph consists of Java 3D objects, called nodes, arranged in a tree structure. The factors that affect efficiency are the *number* and *types* of nodes in the scene graph structure.

All the node objects in a scene graph are derived from the Node class. Java 3D refines the Node object class into two subclasses: Group and Leaf node objects. Group node objects group together one or more child nodes. A group node can point to zero or more children but can have only one parent. Leaf node objects contain the actual definitions of shapes (geometry), lights, sounds, and so forth. A leaf node has no children and only one parent.

Our method comprises two components:

(i) Converting TransformGroup nodes to Group nodes by applying the transforma-

tion in the Geometry node level,

(ii) Combining shapes that have the same appearance into a single Shape3D node.

The first component helps increasing the real time interaction speed while the second component decreases the memory needed by the scene graph structure, thus allowing loading larger molecules.

We explain these two techniques by giving an example of creating a space-fill (van der Waals) model of a protein molecule. The space-fill model consists of spheres of different sizes transformed to the their correct atomic locations according to the 3D atomic coordinates read from the PDB file. The intuitive way to create a space-fill model is to use the Sphere objects provided by the Java 3D API to create spheres of desired size and add them to the TransformGroup objects to translate them to their correct position. Figure 3.8 shows a scene graph structure created by using this method. However, as the number of atoms in a molecule increases the number of TransformGroup nodes increases since each atom has a unique position in the molecule. This makes interaction with the scene very inefficient because at each frame all the TransformGroup nodes need be processed to get the new position of each atom. This process involves a 4x4 matrix multiplication for each TransformGroup object.

To improve on the situation, one observation we made is that the protein molecule is static during interaction, i.e. individual atoms do not move freely. So, according to the interaction's nature **one** TransformGroup node is enough for representing protein molecule's rigid structure's position. However, by using Java 3D's Sphere nodes it is not possible to implement this solution, because the Sphere class does not allow

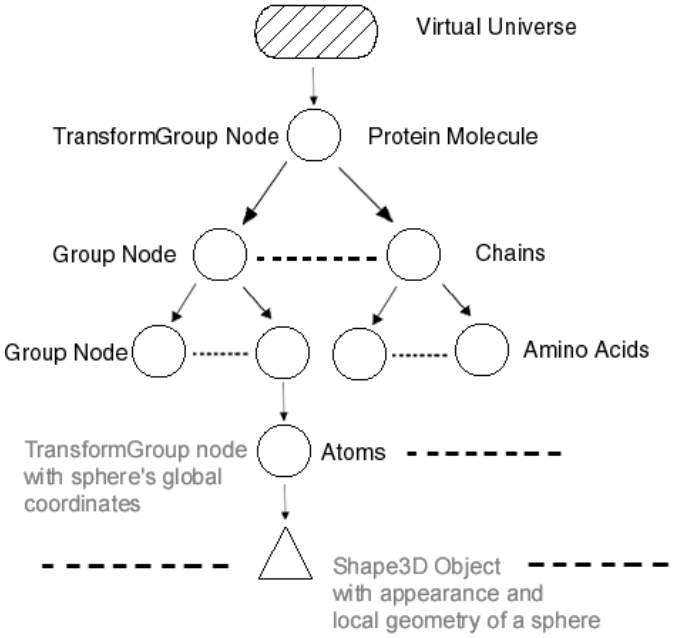

Figure 3.8: A fragment of an intuitive scene graph for the *spacefill* model.

creation of a sphere at an arbitrary position. Thus the only way to create a sphere at a specific position is to put a TransformGroup node above it.

But, there's a way to get around this restriction of Java3D. We have implemented our own Sphere class, which allows a sphere to be built at a specific location. By doing this, what we actually did was to propagate the transformation in the TransformGroup node to the geometry level, by creating geometry at a given static location. This puts a little overhead to the scene building process, i.e. by applying transformations during scene graph creation, but as we show in the next section this overhead is acceptable. The more important thing is that we have reduced the number of Trans-

formGroup nodes in our scene graph to *one* (the one for the whole molecule) by getting rid of *all* the TransformGroup nodes representing individual atoms. As will be shown later, this modification improves the interactive rendering speed significantly. Figure 3.9 shows the scene graph after this improvement.

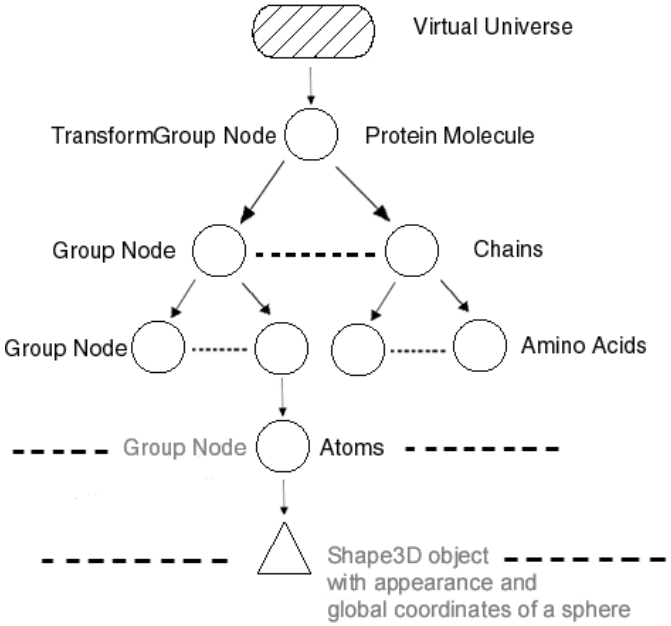

Figure 3.9: The scene graph after applying the first technique.

As seen in Figure 3.9 each sphere is represented by a Shape3D object which encloses its *geometry* and *appearance*. The scene graph contains as many Shape3D objects as the number of atoms in the protein molecule. As the molecule size increases these increasing number of Shape3D nodes may cause memory problems. One way to overcome this is to put spheres with the same appearance under a sin-

gle Shape3D node by combining their geometry information into a single geometry array. The number of Shape3D objects we need is equal to the number of different sphere appearances. For example, if we want to color each atom in a different color, we only need 6 Shape3D nodes, since the protein molecules consist of 6 different atoms (Carbon, Oxygen, Nitrogen, Hydrogen, Sulphur, and Phosphate). This way we can get rid of many Shape3D objects and free up memory space. This technique enables us load very large molecules, which contain as many as 4000 amino acids. Figure 3.10 shows the scene graph after application of this second technique.

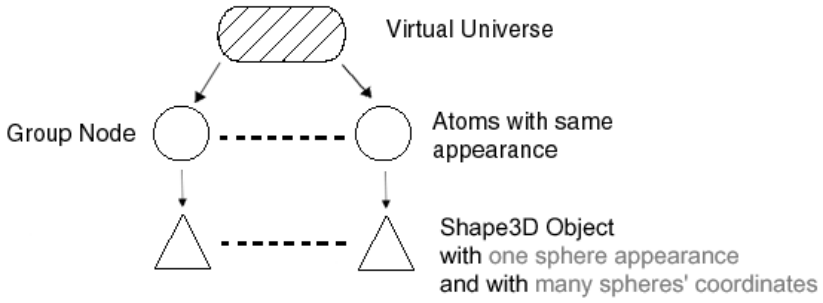

Figure 3.10: The scene graph after applying the second technique.

What we have provided with these techniques is actually a hybrid method combining both retained mode and immediate mode graphics. The immediate mode is simulated by breaking the scene graph hierarchy and collapsing some nodes into a single node to save up memory space and to increase real-time interaction speed. In the next section we demonstrate the effectiveness of our methods by providing some test results.

3.6 Performance Tests and Results

We have compared our system (FPV) to two other molecular visualization tools according to their *scene building* and *real time interaction speed* performances. These tools chosen for the tests (JMV 0.85 and JMVS2) are among the few available molecular visualization tools based on Java 3D. We have chosen JMV and JMVS2 because they are closer to our system in terms of purpose and functionality.

The tests were performed using JAVA2 JRE 1.4.1 and JAVA 3D 1.2.1_04 (DirectX version) on a Microsoft Windows XP machine with Intel Pentium 4 Processor at 2.0GHz and 512MB of RAM. We have dedicated 256MB of this as the maximum size of memory allocation pool for Java Virtual Machine. The graphics accelerator card used for the tests was 64MB DDR NVIDIA GeForce4 MX Graphics Card. The data set comprised 22 protein structures in PDB format ranging in size from 29 amino acids (1bh0) to 8337 amino acids (1aon). Table 3.1 shows the protein molecules and their sizes respectively (both in terms of *number of amino acids* and *number of atoms*).

We have chosen three different types of visual representations to perform the tests: the spacefill (or van der Waals) model, the bonds (or wireframe) model, and the ribbon model. The ribbon model type did not exist in JMVS2 so that part of test was performed on JMV and our system only. The *tube* model type of JMV, which was very close to our ribbon representation, was compared as the ribbon model. We tried to make the visual representations as close as possible by adjusting the display options of the compared systems, e.g. number of sphere divisions. Figures 3.11,

Protein (PDB ID)	*Size (# of residues)*	*Size (# of atoms)*
1bh0	29	242
1ptq	50	402
1df4	68	463
1gcm	102	814
1k52	144	1122
2aid	198	1516
1d9c	242	1993
1a4f	287	2250
3mds	406	3282
1syn	528	4300
1d3a	606	4602
1a05	716	5386
1duv	999	7648
1a0s	1239	9606
13pk	1660	12508
1f8r	1992	15291
1b25	2476	19144
1l1f	3030	23244
1dp0	4092	32500
1h6d	5196	35555
1gyt	6036	46152
1aon	8337	58688

Table 3.1: Sizes of test proteins

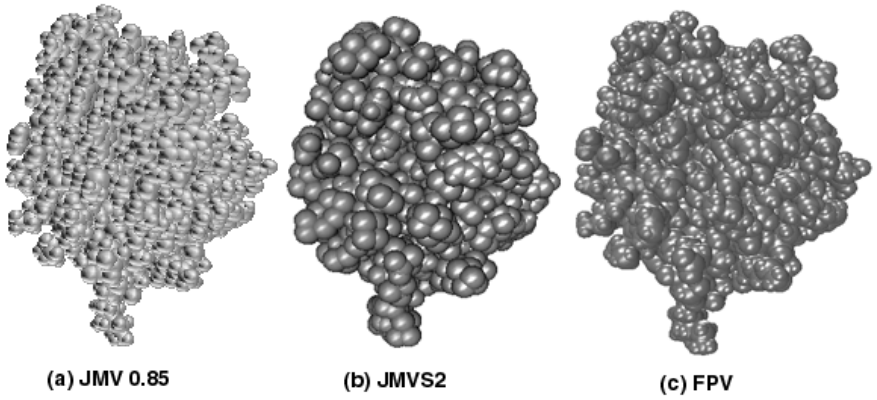

Figure 3.11: The *spacefill* model for the protein molecule 2mhr.

3.12, and 3.13 shows, for each system, the visual representations used for the tests.

The calculation of the timings and rendering speed measurements was possible because source codes of both tools were available. We've measured the scene building times and real-time interaction speed. The scene building times become important, when the user wants to switch between models during interaction. The latency between switching from one representation to another can be intolerable if it is more than a few seconds. One may consider building all the available models during start-up to decrease model switching time during interaction, but this requires much more memory compared to the memory required by a single model type. Therefore, the size of the largest loadable protein molecule decreases drastically. All the programs that we've compared use the suggested approach, which is building a specific model type on demand. That's why we've taken scene building times into consideration. The importance of the real-time interaction speed is obvious. It is one of the main

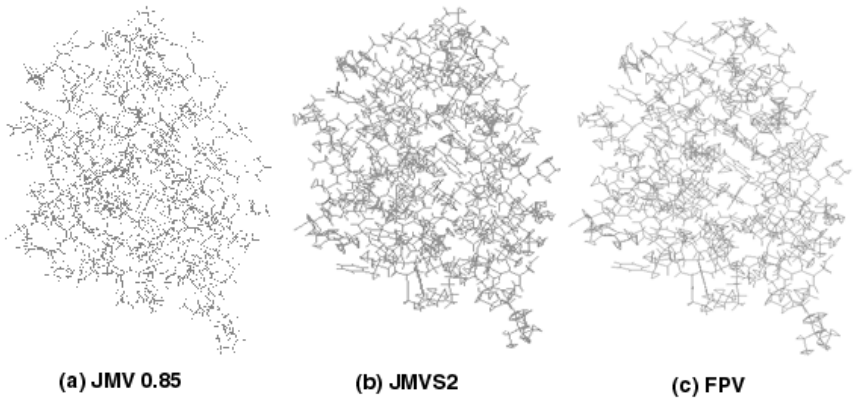

Figure 3.12: The *bonds* model for the protein molecule 2mhr.

quality metrics of interactive visualization tools.

Figures 3.14, 3.16, and 3.17 show results of the rendering speed tests. To measure rendering speed we've used a RotationInterpolator object to have the molecules rotate around y-axis at a constant speed. We then calculated rendering speed by looking at the difference in frame numbers at certain time intervals. Values of 25 and more are ideal in the graphs showing the results of rendering speed tests, because 25fps is the highest frequency the human eye can detect.

In the spacefill model rendering speed test, our system had better performance compared to the other programs, while they performed close to each other. That's because the new Sphere classes that we have implemented to get rid of the TransformGroup nodes and encapsulate many spheres under a single Shape3D node. Thus our system had up to eight times better rendering speed performance (at protein 1duv) compared to the other programs. Furthermore, our system was able to load

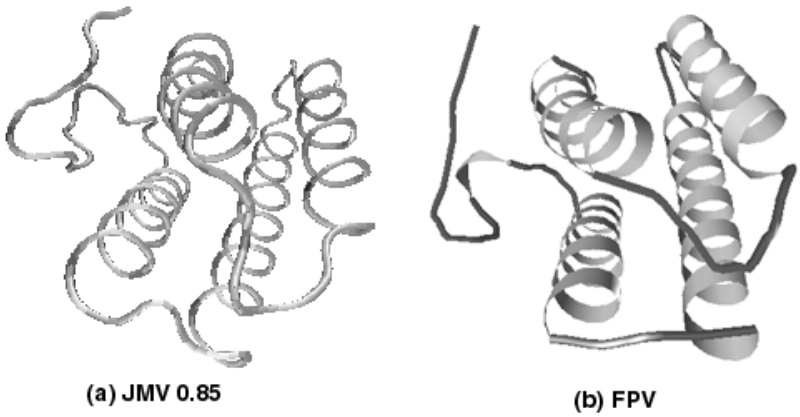

Figure 3.13: The *ribbon* model for the protein molecule 2mhr.

the largest molecule, which has 58688 atoms, while JMV and JMVS2 could at most load proteins that have 35555 and 23244 atoms respectively. Figure 3.15 shows the largest molecule of the test set displayed by our program, FPV. Furthermore, our program could render this molecule at 4 frames per second.

In the Bonds Model test, the performances of our system and JMV were close to each other, while JMVS2 had acceptable speeds for only small molecules. JMV performed better than FPV for large molecules, but it should be noted that even for those large molecules FPV could establish a rendering speed over 26 frames per second. So the difference between JMV and our program was not noticeable practically. Our scene graph structure for the bonds model consists of a single line segments array for all of the bonds of the protein molecule, thus resulting in a very simple scene graph structure. The JMV program uses a similar approach thus has similar performance results. However, the scene graph used by JMVS2 tries to put

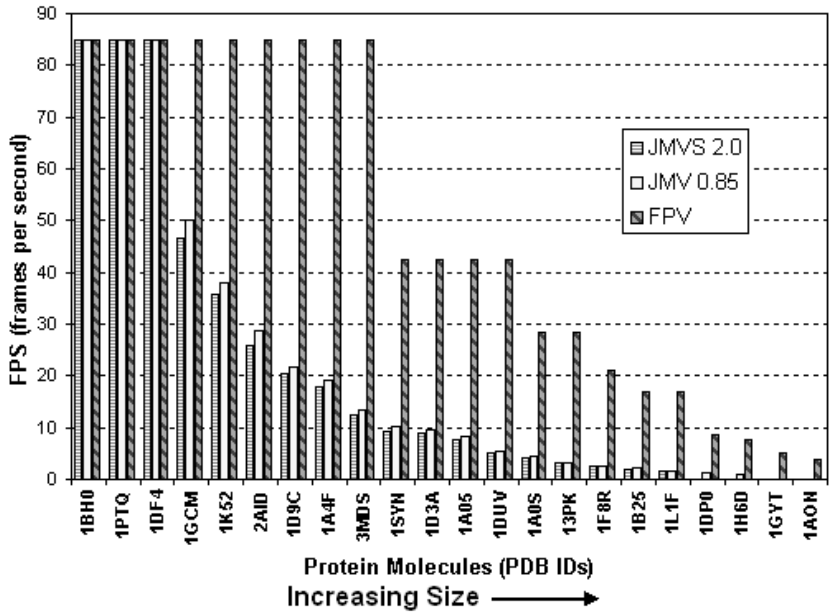

Figure 3.14: Rendering speed for the *spacefill* model.

every bond in a separate Shape3D object, thus resulting in a very poor performance.

The rendering speed comparison of the ribbon model was performed only with the JMV program. Our program performed better than JMV as seen in Figure 3.17. In ribbon model test, we could again load the largest molecule in our data set (1aon), which was 8337 amino acids long, while the largest molecule loaded by JMV had 3030 amino acids (1l1f). Furthermore, FPV achieved up to 20 times better rendering speed performance (at protein 1b25) compared to JMV.The main reason for this was again our method of combining related primitives under a single scene graph node.

Figures 3.18, 3.19, and 3.20 show results of the Java 3D scene graph building tests. By presenting these results we show that the scene graph manipulation tech-

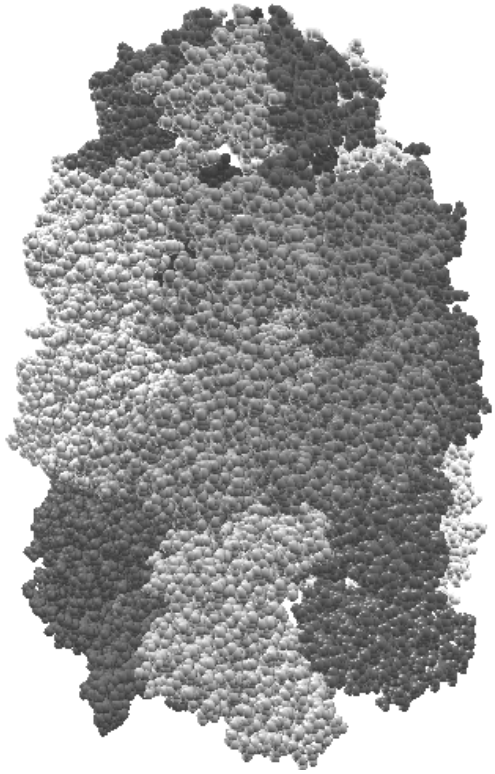

Figure 3.15: The *spacefill* model for the protein 1aon.

niques we've developed do not cause any overhead on molecular scene building.

For the *spacefill* model JMV program performed worst among all three programs compared. The time required to build the molecular scene grows very quickly with the size of the protein. For the large molecules the time required for JMV to build a molecular scene can grow up to hundreds of seconds, which is not acceptable. JMVS2 and our program had reasonable scene building times in the scene graph building tests for the *spacefill* model type.

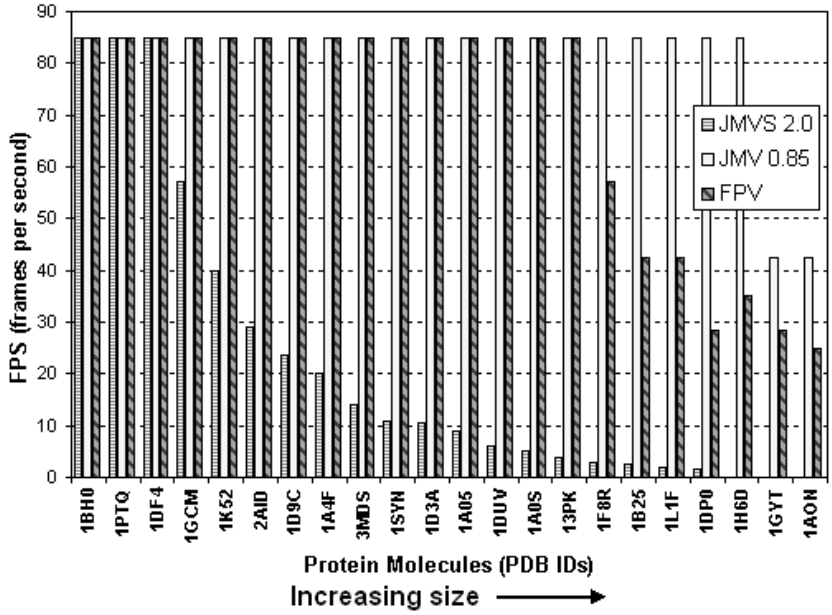

Figure 3.16: Rendering speed for the *bonds* model.

For the bonds model, our program and JMV had similar results and the scene building time was negligible (much less than 1 sec). This time JMVS2 performed poorly compared to our system and JMV. In these results, it is seen that processing some primitives together under a single Shape3D note has benefits rather than an overhead.

Our program outperformed JMV on the scene graph building test for the ribbon model. As mentioned before the ribbon model tests were not performed for JMVS2 because it didn't have a ribbon type representation. It can be seen from Figure 3.20 that scene graph building times for FPV were less than 1 second for all the test proteins. This means FPV has a very low latency when switching between model

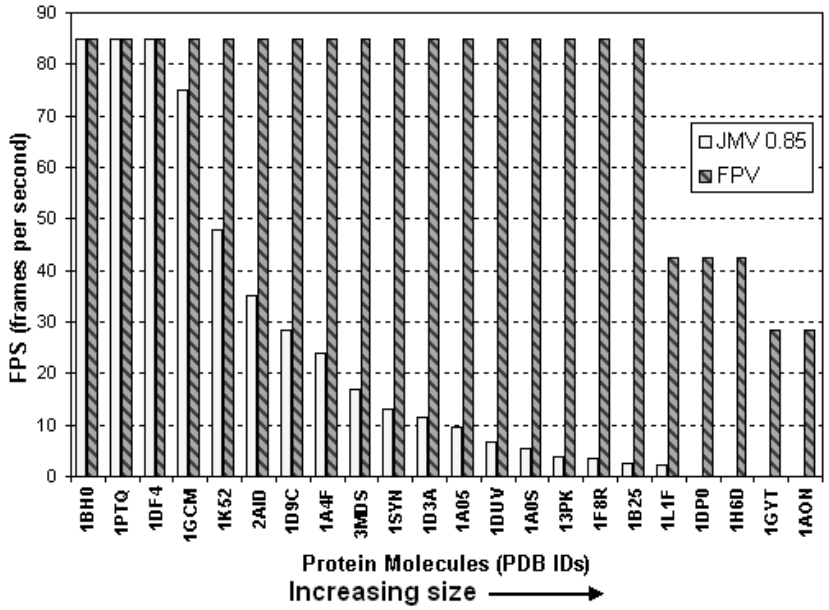

Figure 3.17: Rendering speed for the *ribbon* model.

types during interaction with the molecule. The results for this test again shows that processing related primitives together under a single group node has benefits instead of an overhead during scene building.

3.7 Discussion

In this chapter, we have presented a high-performance protein visualization application called FPV. We've proposed implementation techniques to increase the usability of our application by improving the real-time rendering speed and increasing the range of protein data that can be examined. These improvements are accomplished

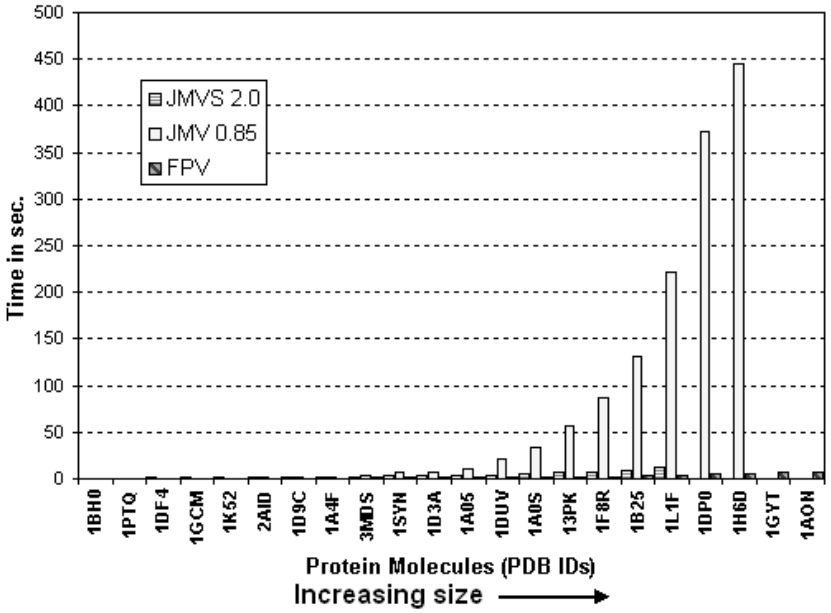

Figure 3.18: Scene building times for the *spacefill* model.

by modifying the scene graph structure used by the Java 3D API. We have showed the effectiveness of our methods by comparing our system to two other molecular visualization tools based on Java 3D.

In order to make our tool more attractive to researchers, we are looking for ways to increase the functionality of our system. One way incorporating new functionality is providing new 3D representation types for protein molecules, such as electron density map and molecular surface representation. Since we've designed the visualization system as a toolkit, it is easy to add new functionalities depending on an application's needs, such as adding superpositioning functionality to the Graphics Module for comparison of protein structures. The design of our system also allows

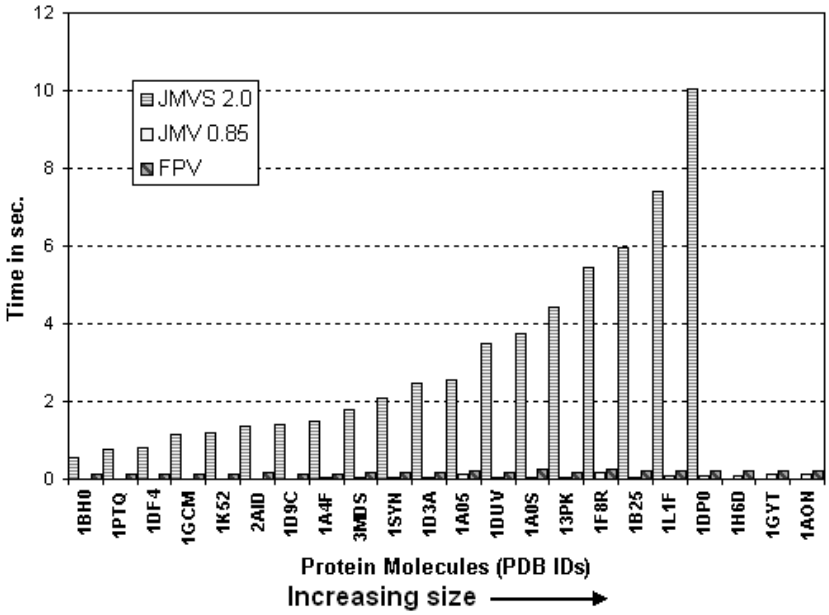

Figure 3.19: Scene building times for the *bonds* model.

users to decouple and use components of the system, such as PDB Loader Module.

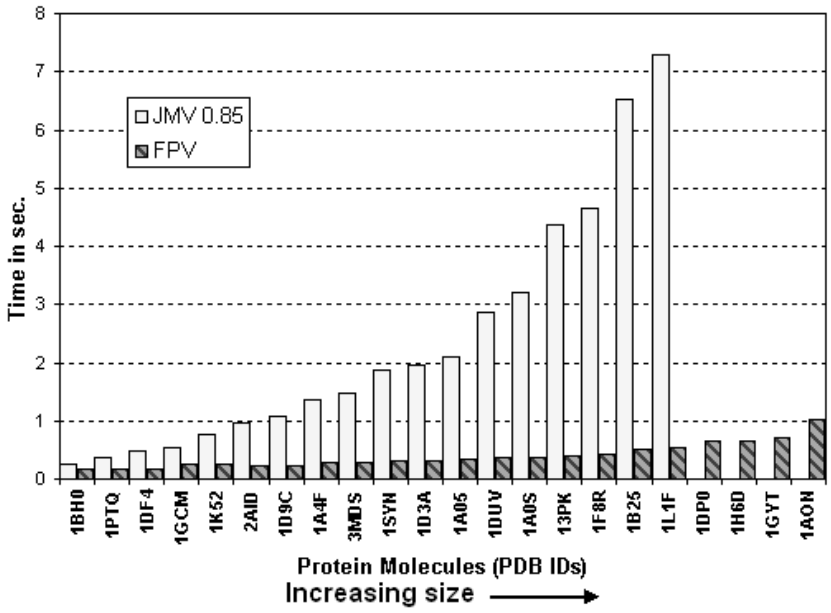

Figure 3.20: Scene building times for the *ribbon* model.

Chapter 4

Methods for Fast Molecular Surface Generation and Interior Cavity Detection

Molecules interact through their surface residues. Calculation of the molecular surface of a protein structure is thus an important step for a detailed functional analysis. One of the main considerations in comparing existing methods for molecular surface computations is their speed. Most of the methods that produce satisfying results for small molecules fail to do so for large complexes. Large complexes such as virus capsids, ribosomes, and chromosomes can contain as many as 100K atoms in their structure data. This increasing complexity of structural data poses a significant problem for analytical molecular surface computation methods since they need to search for all the possible solvent probe placements on the surface by examining every pair, and triple of molecule atoms. In this chapter we present a grid-based approach to compute and visualize a molecular surface at a desired resolution. Our method is based on the emerging level set methods that are used for computing

evolving boundaries in several application areas from fluid mechanics to computer vision. Our method is able to calculate the surface and interior inaccessible cavities very efficiently even for very large molecular complexes. We have compared our method to some of the most widely used molecular visualization tools (Swiss-PDBViewer, PyMol, and Chimera) and our results show that we can calculate and display a molecular surface 1.5 to 3.14 times faster on the average than all three of the compared programs. Furthermore, we demonstrate that our method is able to detect all of the interior inaccessible cavities that can accommodate one or more water molecules.

4.1 Introduction

Interactions between molecules are usually induced by the properties of their surface components. Sequences may diverge and secondary structure arrangements may change topology with the evolutionary process, however surface properties that are essential to protein function are usually conserved. Therefore, calculation and analysis of molecular surfaces play an important role in discovering the functional properties of a protein.

Three main molecular surface definitions exist in the literature[31]. Figure 4.1 shows an illustration of those definitions. The *van der Waals* surface is the area of the volume formed by placing van der Waals spheres at the center of each atom in a molecule. *Solvent-accessible surface* is formed by rolling a solvent, or *probe,* sphere over the van der Waals surface. The trajectory of the *center* of the solvent

sphere defines the solvent-accesible surface. Whereas, the *solvent-excluded surface* is defined as the trajectory of the *boundary* of the solvent sphere in contact with the van der Waals surface. Solvent-excluded surface is usually referred to as *the molecular surface*. Molecular surface and solvent-accessible surface are the most commonly used representations for both graphical visualizations and quantitative calculations of the surface area[31].

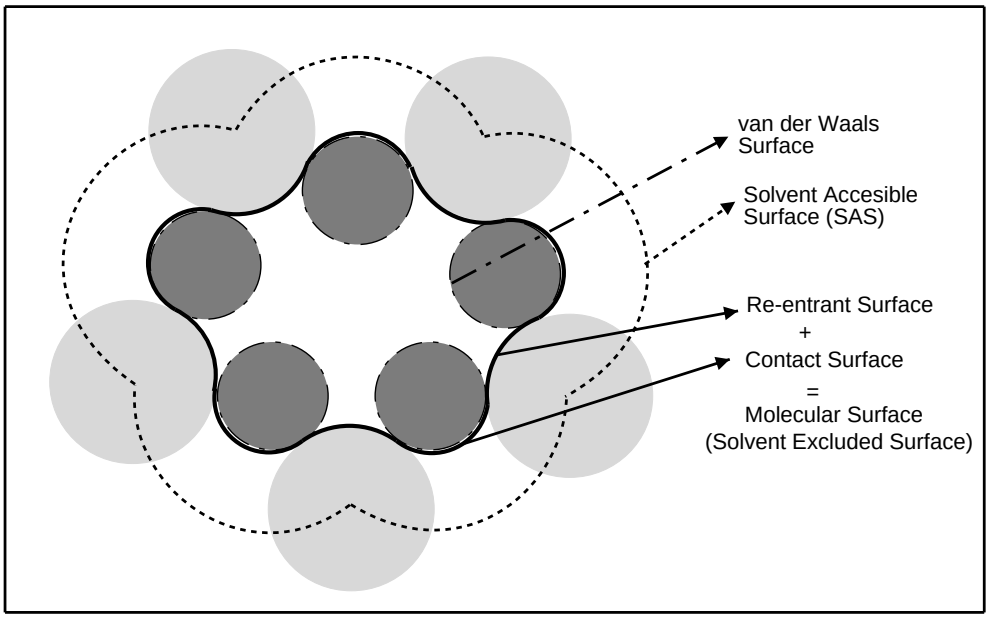

Figure 4.1: A two-dimensional illustration of surface definitions.

Protein molecules are usually well packed. In fact, the packing efficiency of atoms inside proteins is roughly as expected for the close packing of hard spheres[31]. However, Hubbard and Argos[47] analyzed internal packing defects or *cavities* (both empty and water-containing) within protein structures and defined 3 cavity classes:

within domains, between domains, and between protein subunits. These cavities may have several important functions. Takano *et al.* show that buried water molecules in internal cavities contribute to protein stability[86]. Water-filled cavities also play the role of modulating pK_a values of acidic and basic residues surrounding the cavities[55]. Therefore, in the absence of high-resolution structural data capable of resolving all the water molecules inside protein cavities, it is extremely useful to develop accurate and fast computational methods for quantitatively calculating the shapes and sizes of these cavities. The proposed technique addresses both the surface generation and cavity detection problems.

4.1.1 Related work

Numerous methods have been developed to compute molecular surfaces. Here, we describe some of those methods. See [31] for a more thorough review of the area. One of the earliest algorithms was proposed by Connolly [18, 19]. A molecular dot surface is formed as a combination of convex, toroidal, or concave patches when a probe sphere is tangent to one, two, or three atoms respectively.

A grid based algorithm was described by Nicholls *et al.*[66] and used in the program *GRASP*[65]. The method we propose in this chapter is similar to their algorithm except for the detection of the interior cavities. They detect the cavities by choosing a seed point at an extrema, that does not belong to a cavity. All points associated with it, those which can be reached by travelling along triangle edges, are deemed the *noncavity surface*. All others belong to cavities. Note that this will give an incorrect assessment if there is more than one disconnected surface. Unlike

GRASP, our method can handle such topologies naturally without any effort. *GRASP* is currently available only for SGI machines, therefore it is not among the programs that we used for comparison.

Sanner *et al.*[74] developed a method that relies on the reduced surfaces for computing the molecular surfaces. The reduced surface corresponds to the alpha shape[25] for that molecule with a probe radius α. An implementation of this method (MSMS package) is used by the UCSF Chimera[45] molecular graphics program, which is one of the programs that we compared our method to.

All of the described methods work very well for small molecules. However, one may need to analyze large complexes as the interaction of proteins with DNA and RNA is essential for many cellular functions. Therefore, development of a method that is capable of interactively analyzing and visualizing the molecular surface representations of these large complexes is extremely important.

4.1.2 An overview of our method

We use a grid based approach to compute the molecular surface of a protein with known structure. Our method, which we name LSMS: **L**evel **S**et method for **M**olecular **S**urface generation, proceeds in three stages:

1. Mark grid cells that are inside the solvent-accessible surface.
2. Mark grid cells that are outside the solvent-excluded surface.
3. Use the fast marching level set method to determine the outer surface and interior cavities of the molecule.

The volume and area calculations of the molecular surface as well as the internal inaccessible cavities is then carried out very efficiently on the processed grid. For visualizing the molecular surface, a triangular mesh is generated using the marching cubes method[57].

We have evaluated LSMS for generating molecular surfaces of very large molecular structures having 27375 (PDB id: 1a8r) to 97872 (PDB id: 1hto) atoms. We compared our results to PyMol[21], Chimera[45], and Swiss-PDBViewer[37] and our results show that LSMS is faster than all of those tools. We have also performed experiments to evaluate the extent of LSMS's interior cavity detection capabilities. We compare the internal cavities found by LSMS to the cavities found by Swiss-PDBViewer. Our results show that LSMS can find all the cavities that can accommodate one or more water molecules. Hence, our technique makes two significant contributions: (1) time and memory efficient mechanisms for computing and visualizing molecular surfaces and (2) accurate determination of interior cavities.

The rest of the chapter is organized as follows. We present the details of our technique in Section 3.2. We present the experimental results of molecular surface performance tests and interior cavity detection tests in Section 3.3. We conclude with a brief discussion in Section 3.4.

4.2 Methods

The input to our method is the atomic coordinates of the molecular structures as a PDB[8] file. We ignore the hydrogen atoms during the surface computation and

employ the commonly used *united atom* approach[31]. In this approach, the size of an atom is enlarged by accounting for its hydrogens. We use the same united atom radii applied in Rasmol's[77] spacefill rendering[1].

The molecular structure is then placed and centered on a three-dimensional orthogonal grid of a desired resolution. The size and the resolution of the grid is the same along all of the three dimensions and constant during molecular surface computations. The resolution of the grid with the size of the molecule defines a quality measure that directly corresponds to the quality measure employed by Swiss-PDBViewer[37], that is the number of grid cells per 1.4 Å. We resize the molecule uniformly in all dimensions so that it fits completely inside the cubic grid. The quality is therefore given by $(N/L) \times 1.4$, where N is the resolution and L is the length of the molecule (in Å) along the major axis.

A molecular surface for the input structure is computed in three stages. First, we mark grid cells that are inside the solvent-accessible surface of the molecule. Then, the grid cells that are outside the solvent-excluded surface is marked as outside. However, at the end of the second stage the cavities inside the molecule are also marked as outside cells. The surface surrounding those cavities is not distinguished from the outside molecular surface that is accessible to solvent molecules. Therefore, we use the level set method to distinguish the outer surface from the interior cavities, by shrinking a boundary that initially encloses all of the molecule.

The volume and area calculations of the molecular surface as well as the internal inaccessible cavities is then carried out very efficiently on the final processed

[1] http://www.umass.edu/microbio/rasmol/rasbonds.htm

grid. For visualizing the molecular surface, we generate a triangular mesh using the marching cubes method[57]. The details of the above steps are explained in the following subsections.

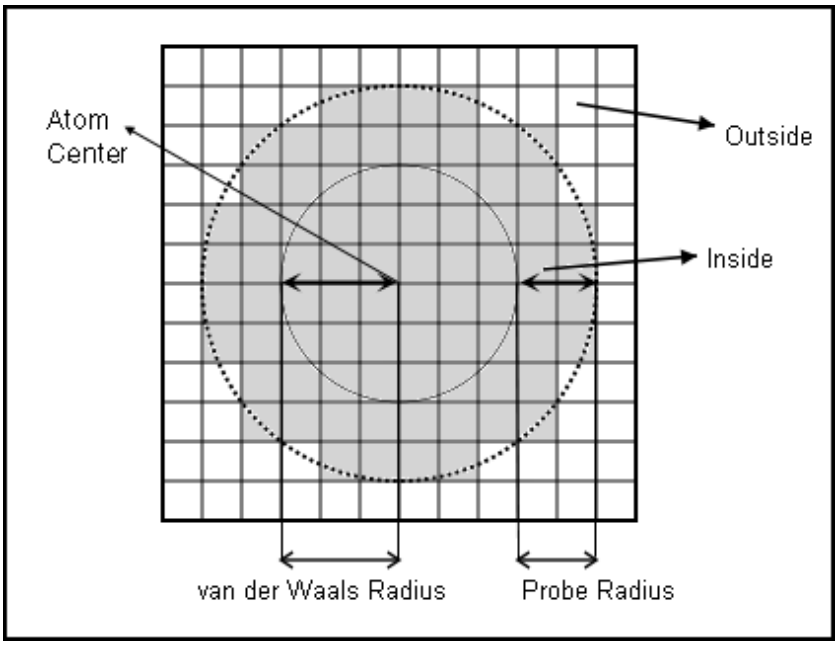

Figure 4.2: The grid cells whose centers fall inside the volume defined by the solvent-accessible surface is marked *inside*.

4.2.1 Marking the volume inside the solvent-accessible surface

The first stage in the molecular surface computation consists of marking the grid cells inside the volume defined by the solvent-accessible surface. We proceed as follows. By default all the grid cells are considered outside of the surface. Then, by traversing all of the atoms of the molecular structure we mark the cells, whose

centers fall inside the volume defined by the solvent-accessible surface, as inside. Figure 4.2 illustrates the process in two dimensions for one of the atoms of the molecule. The extension to three dimensions involves spheres instead of circles.

There are a couple of points to consider in terms of the computational complexity of the procedure described above. The marking of the grid cells that are inside takes $O(m \cdot k)$ time, where m is the number of atoms and k is the average number of grid cells occupied by an atom. Note that this process may visit the same grid cell more than once, since the enlarged van der Waals volume of the atoms may intersect. Another approach to overcome this repetition may be traversing the grid cells in one pass and checking if they are inside an atom or not. The brute force implementation of this technique will take $O(N^3 \cdot m)$ time in the worst case, where N is the grid resolution along one dimension and each grid cell is checked for intersection with every atom. We can optimize this by building an octree data structure over the atoms of the molecule. This will reduce the time complexity to $O(N^3 \cdot \log m)$. However, our experiments on a 256×256×256 grid showed that even with this optimization the second approach is about 6 times slower than the first approach, that has $O(m \cdot k)$ time complexity. It takes 11.34 seconds to process the protein 1pma (45892 atoms), on a 256×256×256 grid using the octree approach, however, it only takes 1.86 seconds when the molecule traversing approach is used.

4.2.2 Finding the solvent-excluded surface

After marking the grid cells inside the volume of the solvent-accessible surface, the next step is to mark out the parts that fall inside the solvent molecule, hence the

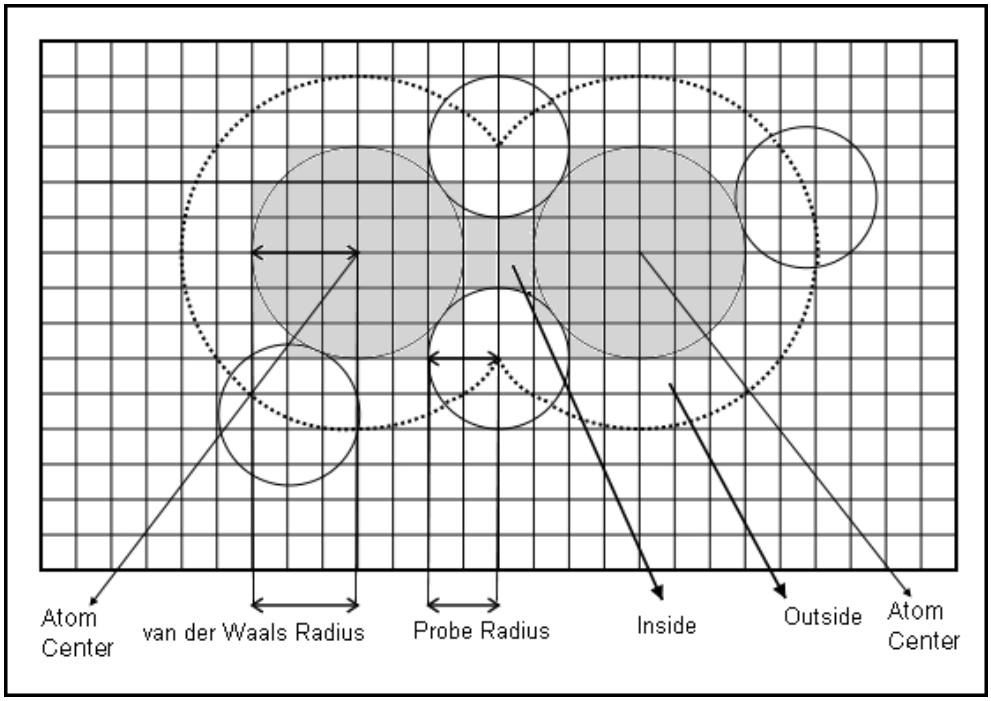

Figure 4.3: The grid cells whose centers fall inside the probe circles are marked *outside*.

name, *solvent-excluded surface*. This process is the most computationally expensive stage of our method. It involves finding all the probe spheres centered on the solvent-accessible surface. Then, for each such probe, the grid cells that are inside the probe sphere is marked as outside of the molecular surface. Figure 4.3 illustrates this procedure in two dimensions. The grid cells that are marked as inside in the previous stage are now marked outside by this procedure, if they fall inside a probe circle. Again, the extension of this illustration to three dimensions involve probe spheres instead of probe circles.

The time complexity of the second stage depends on the resolution of the grid

as well as the complexity of the solvent-accessible surface. A protein surface which contains a lot of pockets (outside cavities) will require a larger number of probes compared to a surface that is smooth. Therefore, it would require more computation time. The complexity of this stage is $O(P \cdot k)$, where P is the number of possible placements of the solvent (probe) molecule, and k is the number of grid cells occupied by it.

4.2.3 Interior cavity detection using fast marching level set method

The result of the first two stages is a grid that represents the volume occupied by solvent-excluded surface of the molecular structure. However, the interior inaccessible cavities is not distinguished from the surrounding space in this representation. As a result, the volume occupied by them is excluded in the total molecular volume. Furthermore, if one computes the molecular surface area using that grid, the resulting surface area will not be the area of the molecule in contact with its surrounding environment, but it will also include the surface areas of the interior inaccessible cavities. Therefore, we need to distinguish between the outer surface and the interior cavities. Figure 4.4 illustrates an inaccessible cavity in two dimensions. Note that, in three dimensions the illustration should not be realized as a torus shape, but instead as a small sphere inside a larger sphere, where the small one would be the inaccessible cavity.

We propose a method based on the level set method to solve this problem. The level set method[79, 69] provides a mathematical framework to compute evolving boundaries. It is based on a continuous formulation usually by partial differential

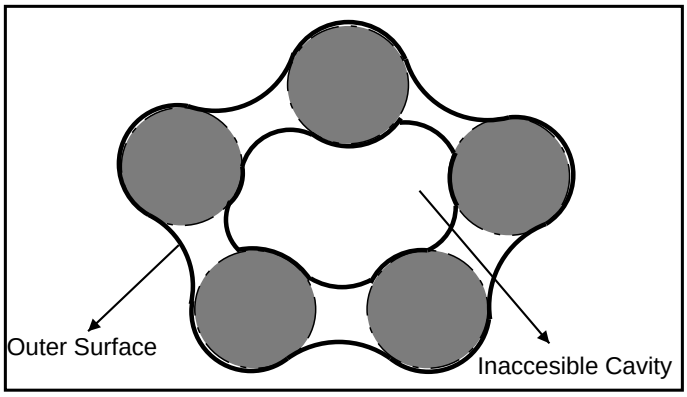

Figure 4.4: A two-dimensional illustration of an inaccessible cavity.

equations and allows one to deform an implicit surface, which is usually the zero iso-contour of a scalar (level set) function. The topological changes, e.g., split, merge, are handled naturally by the level set method. The level set formulation works in any number of dimensions and the computation can easily be restricted to a narrow band near the zero level set for efficiency.

The idea is to initialize a boundary that encloses the molecule, then shrink the surface at a constant speed. The stopping criterion in the speed function is the encounter with a grid cell that is marked as inside. The evolution of the boundary stops completely when all of the boundary points are stopped by an inside grid cell. The key observation here is that during this shrinkage process, the boundary has a fixed signed speed, i.e., a grid cell passed over by the boundary will not be visited again by any part of the boundary.

With this observation we can apply the fast marching method, in which the closest grid points to the boundary are considered first, and a grid cell that is processed

is never processed again. In this procedure we maintain a narrow band of grid cells that represents the current boundary, and update that narrow band as the boundary evolves. At the end of this procedure the outer surface is detected, and the voids inside the molecular surface are detected as interior inaccessible cavities. We present examples of interior cavities in the experimental results section.

4.3 Experimental Results

We have conducted two sets of experiments to evaluate the performance and utility of our method. First, we evaluate the performance of LSMS for computing and visualizing molecular surfaces of very large complexes. Then, we compare the cavity detection accuracy by comparing our results to cavities reported by Swiss-PDBViewer. We explain the experiments in detail in the following subsections.

4.3.1 Molecular surface generation and visualization performance

Computing and visualizing the solvent-excluded surface of a molecular structure is a computationally challenging task. Most of the existing visualization tools achieve similar performance for small molecules despite the difference of underlying methods used. However, not all of the methods can cope with the increasing size of the molecules. Therefore, we have selected a challenging set of large molecular structures as a benchmark dataset. The dataset consists of 15 large complexes that contain 27375 to 97872 atoms in their structure data.

We have compared LSMS's molecular surface generation performance to three

		surface generation time (sec.)				surface quality	
Protein	size	LSMS	SPDBV	PyMol	Chimera	LSMS	SPDBV
1a8r	27375	11.66	14.96	51.34	16.36	1.03	1
1h2i	32802	15.66	17.33	40.78	40.04	1.29	1
1fka	34977	37.14	51.56	85.14	77.25	1.34	1
1gtp	35060	15.81	19.75	50.17	67.04	1.28	1
1gav	43335	20.31	35.24	86.62	78.35	1.28	1
1g3i	45528	26.80	37.51	63.90	*u*	1.41	1
1pma	45892	40.78	*u*	51.10	*u*	1.67	1
1gt7	46180	14.91	22.60	57.75	54.39	1.16	1
1fjg	51995	30.34	48.44	85.79	*u*	1.33	1
1aon	58884	47.28	61.20	95.84	*u*	1.41	1
1j0b	60948	18.28	44.97	100.14	*u*	1.18	1
1ffk	64281	69.83	72.07	135.80	196.65	1.27	1
1otz	68620	42.05	51.27	78.05	*u*	1.45	1
1ir2	87087	21.09	*u*	120.52	93.87	1.23	1
1hto	97872	38.95	89.68	*u*	*u*	1.28	1

Table 4.1: Molecular surface generation times for LSMS compared to those of Swiss-PDBViewer, PyMol, and Chimera.

other programs that are widely used in the computational biology community and are freely available. PyMOL[21] is an open source molecular graphics system with an embedded Python[73] interpreter designed for real-time visualization and rapid generation of high-quality molecular graphics images and animations. UCSF Chimera[45] is another tool implemented in Python. The solvent-excluded molecular surfaces produced by Chimera are created with the help of the MSMS package[74]. Swiss-PDBViewer[37], or SPDBV for short, is another molecular viewer with extended functionality. There is little amount of documentation about the molecular surface component of SPDBV. Nevertheless, it can be understood from the documentation that the surface computation is carried out on an orthogonal grid as in LSMS. The probe size is 1.4 Å as in other methods, however all the molecule atoms have a fixed radius. The current version of SPDBV does not allow changing of these parameters. The only value that can be altered is the smoothness (quality) parameter. By default it is 1, which means 1 grid point every 1.4 Å. This quality should be enough for most purposes as also indicated by SPDBV's developers[2].

Table 4.3 shows the molecular surface generation times. All of the tests are performed on a Microsoft Windows XP machine with Intel Pentium 4 Processor at 2.0GHz and 512MB of RAM. The results we report here use the programs' default parameter sets and do not include the time taken to load the molecule into memory. The timings for Swiss-PDBViewer are acquired using a quality value of 1. We also compute and report a quality measure for LSMS that directly corresponds to SPDBV's quality measure, i.e., number of grid cells per 1.4 Å. Protein sizes in

[2] http://us.expasy.org/spdbv/text/surface.htm

the table are shown as number of atoms. The entry *u* in the table means that the program is not able to generate a molecular surface for that protein. We have used a 256×256×256 resolution grid for timing LSMS. The quality of the surface is affected by the size of the molecule as well as the resolution of the grid. We resize the molecule uniformly in all dimensions so that it fits completely inside the cubic grid. The quality is therefore given by $(N/L) \times 1.4$, where N is the resolution and L is the length of the molecule along the major axis. Figure 4.5 shows the largest protein in our dataset in 256×256×256 resolution and a quality of 1.28. The boundaries of the 256×256×256 resolution grid is also shown. LSMS can interactively render the surface of 1hto with 9 frames per second display rate.

Table 4.3 shows that LSMS is up to 2.46 times faster than SPDBV (achieved at protein 1j0b) and is 1.5 times faster on the average, while achieving a better quality for every test protein. LSMS is also 3.14 times faster than both PyMol and Chimera on the average. Also, it is important to note that for some of the test cases SPDBV, PyMol, and Chimera are not even able to generate the molecular surface, whereas LSMS successfully computes the surfaces for all of the test cases.

4.3.2 Interior cavity detection

The outer boundary of the solvent accessible surface is found by shrinking an initial enclosing boundary at a constant speed with the fast marching level set method. Figure 4.6 shows such an outer surface of the protein 2ptn computed and displayed by LSMS. However, as we have stated earlier there may exist inaccessible cavities inside the molecular surface that are not visible, i.e., occluded by the molecular

	# of cavities		cavity volume (Å^3)		molecule volume (Å^3)	
Protein	LSMS	SPDBV	LSMS	SPDBV	LSMS	SPDBV
1eca	1	1	31.46	134	16688.98	16824
2act	7	2	322.33	281	6842.76	6509
2cha	10	4	338.01	436	28728.13	27705
2lyz	3	2	96.12	162	15778.44	15133
2ptn	6	3	394.41	380	27037.94	25897
5mbn	4	2	127.05	189	19796.24	19768
8tln	14	2	356.28	170	40280.45	38498

Table 4.2: Cavities computed using LSMS and comparison with results from Swiss-PDBViewer.

surface. Nevertheless, analysis of these cavities may be required to study the buried water molecules inside them which may contribute to protein folding stability.

Figure 4.7 shows the internal cavities of the same protein 2ptn that can accommodate one or more water molecules. The C_α trace is also shown along with the cavities to provide visual clues of relative locations of the cavities inside the molecule.

We have analyzed internal cavities of a set of seven protein molecules. We compared those results to the results reported by Swiss-PDBViewer. Table 4.3.1 shows the results of the cavity detection experiment. We have examined the number of separate cavities as well as the total cavity volume in cubic angstroms found by LSMS and SPDBV. The volume occupied by the molecular surface is also shown to give an idea about the size of the proteins. The results show that LSMS can find all the cavities found by SPDBV. We have also verified these results by further visual inspection. LSMS usually finds more cavities compared to SPDBV. The disagreements in number of cavities and total volume of cavities are probably caused by

the employment of different van der Waals radii by the two methods, as there is no unique established standard for those, as well as SPDBV's fixed atom radii strategy.

4.4 Discussion

In this chapter we have presented a method to calculate the solvent-excluded surface as well as the interior inaccessible cavities of a molecular structure. Our method is based on a fast marching level set method that efficiently formulates a constant signed speed evolving boundary. We have shown that our method, LSMS, is able to calculate the surface and the cavities very efficiently even for very large complexes. The experimental comparison of LSMS to some of the most widely used molecular visualization tools shows that we can calculate and display a molecular surface faster than all three of the compared programs. LSMS can also detect all the cavities that can accommodate one or more water molecules.

A future research direction based on our work described in this chapter is the development of methods for computation of molecular surfaces dynamically as the probe radius changes. Starting on the van der Waals volume and using distance-transform one can generate a list of initial surface points (zero distance) and then propagate from those points outward to a particular distance, k (the maximum probe radius). The solvent-accessible surface for a particular probe radius $r < k$ is then readily computed by this method. The solvent-excluded surface can also be determined dynamically by shrinking the accessible surface using another distance transform (fixed-speed level set formulation). All the points that are r distance from the

initial surface will give the solvent-excluded surface. This two-step level set method can be used to adjust the probe radius dynamically.

An application of molecular surface computations is the comparison of generated surfaces. Molecular surface comparison is a more difficult computational challenge compared to sequence or structure comparison methods. Finding surface similarities among a family of proteins may help reveal the functional determinant of that family, and the other dual problem of finding a complementary surface (the docking problem) may help in drug discovery and development. Several surface properties can be considered when comparing protein surfaces: electrostatic potential, surface curvature, cavity size, molecular surface area, and molecular volume.

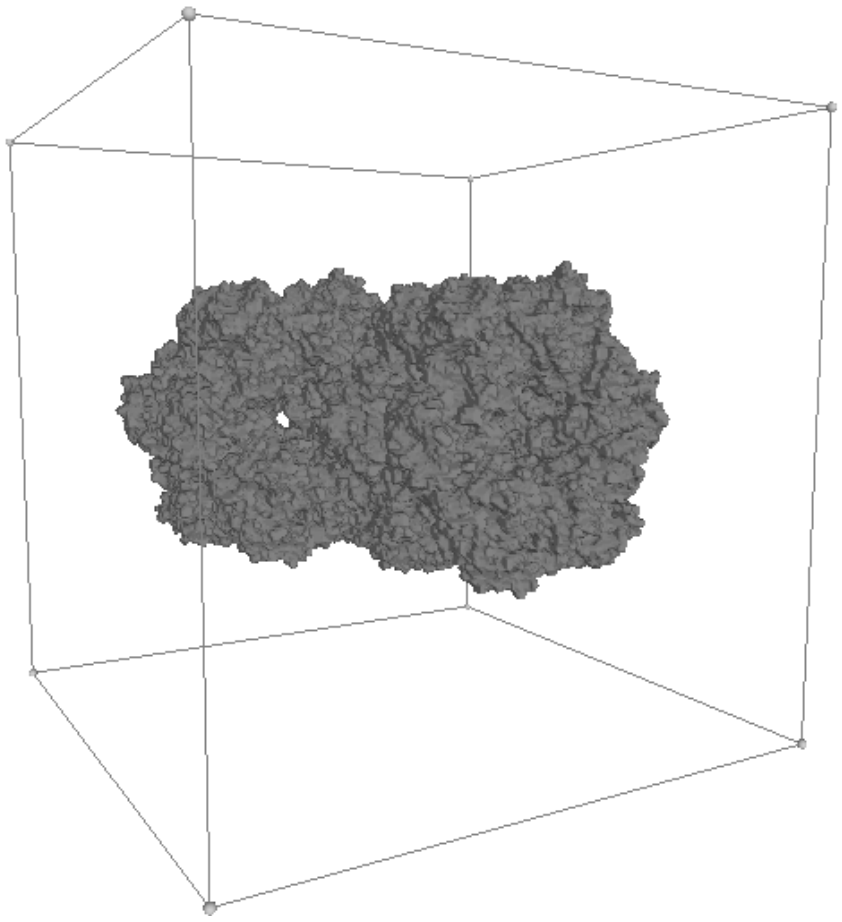

Figure 4.5: The molecular surface of 1hto generated by LSMS.

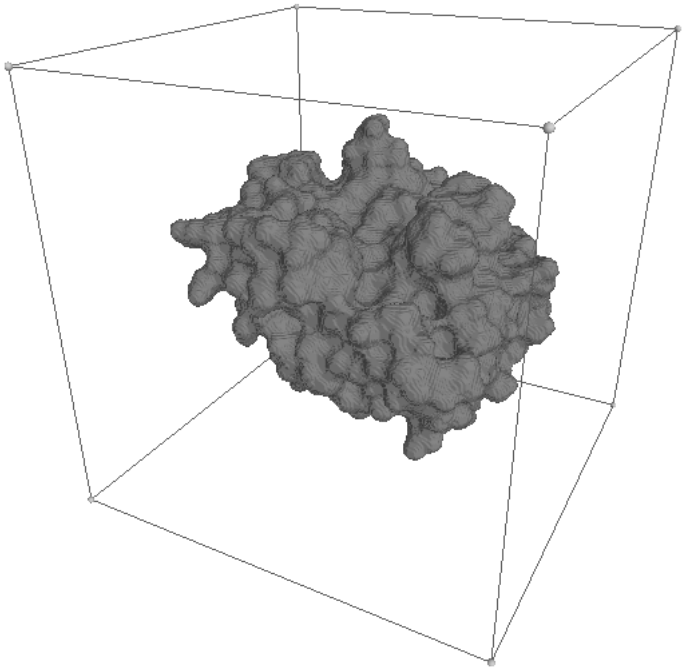

Figure 4.6: The molecular surface of 2ptn generated by LSMS.

Figure 4.7: The inaccessible cavities inside 2ptn along with its C_α trace.

Chapter 5

A Robust and Efficient Algorithm for Protein Structure Similarity Search

In this chapter we present a new method for conducting protein structure similarity searches, which improves on the **efficiency** of some existing techniques. As the number of known protein structures is increasing at a considerably high rate, significant questions arise about the ability of existing methods in handling structure similarity search in a reasonable amount of time. Most of the existing methods [80, 42, 60] for protein structure comparison are designed for pairwise comparison. Therefore, in order to conduct a similarity search of a protein structure against a database of protein structure, one needs to perform an exhaustive search by pairwise comparing the query protein to all of the database proteins one by one. Clearly, this is not a reasonable approach, and methods for conducting fast similarity searches should be developed.

The method we propose in this chapter is grounded in the theory of differential geometry on 3D space curve matching. We generate shape signatures for proteins

that are *invariant*, *localized*, *robust*, *compact*, and *biologically meaningful*. The invariancy of the shape signatures allows us to improve similarity searching efficiency by adopting a hierarchical coarse-to-fine strategy. We index the shape signatures using an efficient hashing-based technique. With the help of this technique we screen out unlikely candidates and perform detailed pairwise alignments only for a small number of candidates that survive the screening process. Contrary to other hashing based techniques, our technique employs domain specific information (not just geometric information) in constructing the hash key, and hence, is more tuned to the domain of biology. Furthermore, the invariancy, localization, and compactness of the shape signatures allow us to utilize a well-known local sequence alignment algorithm for aligning two protein structures. One measure of the efficacy of the proposed technique is that we were able to perform structure alignment queries 36 times faster (on the average) than a well-known method while keeping the quality of the query results at an approximately similar level.

5.1 Introduction

The number of known protein structures is increasing rapidly, as more researchers are joining the hunt for novel protein structures, more experimental apparatus are deployed, and more theoretical frameworks and software tools are developed for predicting protein structures. Protein structure comparison tools play an important role in this enterprise. In predicting a protein structure from its sequence, researchers usually form a new candidate structure. To avoid potential exponential explosion of

structures, that new structure is compared with previously known structures for verification/tuning/correction. Discovering similar folds or similar substructures thus provides restrictions on the conformational space and serves as a starting point for producing useful models [11].

Structure comparison is an NP-Hard problem [53]. There are no fast structural alignment algorithms that can guarantee optimality within any given similarity measure. Therefore, existing structure comparison methods employ heuristics. There are different approaches for extracting structural features. Some methods use only the coordinates of the C_α atoms [42, 80]. They infer the global structure by examining the inter-atomic distances between residues. There are also quite a few methods that use secondary structure elements (SSEs) to simplify the problem by finding initial alignments of SSEs [33, 43, 58, 83, 91] to guide the match of amino acids.

Some methods rely on localized features. In Leibowitz *et al.* [54], a feature extraction method was proposed that examines the k-tuples of atoms in a spherical shell neighborhood of a residue. For retrieving similar structures, geometric hashing is used, which was first introduced in computer vision [52]. Geometric hashing is also used by Nussinov *et al.* [67] and Pennec *et al.* [70]. However, those techniques are not localized as Leibowitz *et al.* [54], and can be slow due to the large amount of redundant information kept. It can take as much as 18 seconds to compare two proteins [70]. Nevertheless, geometric hashing is the first approach that targets the need of indexing for fast similarity searches in a large structure database. Another advantage of geometric hashing is that it can also be used for multiple structural alignment [54]. However, one complaint about the pure geometrical methods is

that since they do not make use of domain specific knowledge, they may overlook some biologically significant relationships such as secondary structure assignment or residue properties, e.g., hydrophobicity.

Another difficulty in the structural comparison problem is the choice of a measure to quantify the similarity between compared structures [35]. One of the widely used measures is the RMSD (root mean square distance) measure [26]. It is a measure of similarity based on the closeness of corresponding C_α atoms of two protein structures. However, a match may involve only a subset of all C_α atoms. I.e., there may exist biologically significant local alignments (even when the molecules do not share a global structural similarity). So the length of the alignment becomes an important measure. The information on the gaps in the alignment also gives hints on the quality of the alignment [26]. Some methods also compute the *p-value*, *e-value*, or *z-value* to quantify the statistical significance of the match [33, 42, 80]. In this chapter, we do not address the problem of finding a universal similarity measure, instead we employ a measure used by Gerstein and Levitt [30], which involves the RMSD and the length of an alignment.

The many variations of protein structure comparison algorithms briefly surveyed above show us that the problem of structure comparison is indeed hard. Furthermore, most of the algorithms are for pairwise comparison. I.e., they need to perform an exhaustive sequential scan of a structure database to find similar structures to a target query protein. This approach may not be feasible as the structure databases, such as the PDB [8], grow in size. Thus, fast and accurate methods for conducting structure similarity searches are needed (there are some efforts in designing methods that

utilize indexing to make similarity searches more efficient, e.g., indexing DALI's distance matrices [6]).

In this chapter, we present a new method for protein structure similarity search and alignment. *The main contribution is to improve on the efficiency of similarity searches. The result is that our method is able to find, efficiently, meaningful structural similarities in proteins about 30 times faster than CE [80], a widely used method for conducting protein structure alignment. We were also able to find similarities that were overlooked by other existing techniques*. Salient features of our method are that we construct signatures for structural matching that are *invariant* (i.e., they are not affected by the translation and rotation of a protein structure in space), *localized* (i.e., the signature at each residue location is completely determined by the local structure around that particular residue), *robust* (i.e., small perturbations of atomic coordinates induce small changes in the associated signatures), *compact* (i.e., the size of the signatures is $O(n)$, n: number of residues), and *biologically meaningful* (i.e., we incorporate secondary structure assignment into the signature). These signatures are constructed and indexed off-line to improve query efficiency. The on-line matching process is carried out in a coarse-to-fine hierarchical manner to enable fast protein structure similarity search and detailed pairwise alignment that handles alignments with gaps. We have implemented our method as an interactive tool that allows visual inspection of the alignment results and iterative discovery of possible suboptimal alignments that may have biological importance.

In the following sections we describe in detail our method. In Section 4.4 we present experimental results. And finally we conclude with future directions and

discussions.

5.2 Methods

Our method is grounded in the theory of differential geometry on 3D space curve matching. The idea of representing protein structures by using differential geometry was first introduced by Rackovsky and Scheraga [72]. However, their work focuses on investigating local curvature and torsion differences between very similar proteins and relating curvature and torsion values to secondary structure conformations, such as alpha helices and beta sheets. On the other hand, our work focuses on finding large scale structural alignments between two protein structures by using curvature and torsion values.

It is well established in differential geometry [22] that the necessary and sufficient condition for structure isomorphism of two space curves is the correspondence of their curvature and torsion values, expressed as a function of the intrinsic arc length. Intrinsic arc length (s) satisfies the property that $|\dot{\mathcal{C}}(s)| = |d\mathcal{C}(s)/ds| = 1$, where $\mathcal{C}$ denotes the space curve. Such a parametrization is in general difficult to obtain in real world applications. However, for protein structure matching, the C_α atoms along the backbone can be considered equally spaced because of the consistency in chemical bond formation. Hence, we can use the polygonal arc length between C_α atoms as a convenient parameterization without loss of generality. Note that the fundamental theorem of differential geometry does not apply to *gapped* similarities. However, ungapped local isomorphism can still be detected and by using

dynamic programming, the matching local substructures can be connected to allow for a larger structural alignment with gaps.

Because of the limited resolution of the apparatus used and noise inherent in any measurement process, the atom positions of a protein structure are imprecisely specified. In order to have robust and reliable shape signatures, smoothing of data points is needed to cope with experimentation and resolution related errors. Approximation splines are used to smooth data points [38, 49]. Furthermore, we use variable error estimates for smoothing different type of secondary structures. This is biologically meaningful, because certain secondary structures (like *turns*) are much more likely to have errors in them.

After smoothing the C_α coordinates of a protein with a polynomial spline, we compute its shape signature. The shape signature of a protein is a list of signature triplets, one for each of its residues. A signature triplet of a residue consists of its secondary structure assignment and curvature and torsion values at its C_α position. Thus, we name our method as CTSS, which is the abbreviation for **C**urvature, **T**orsion, and **S**econdary **S**tructure. These signatures are rotation and translation invariant. In other words if two different curves produce similar curvature and torsion values, then it can be concluded that they are similar (modulo rotation and translation) [22]. Curvature and torsion at a point along the curve provide localized geometrical information. The smoothing process produces a stable signature that is robust in the presence of measurement noise. Furthermore, our method is not purely geometrical because we incorporate biological information such as the secondary structure assignment into the shape signatures. Thus, we achieve stability and ro-

bustness in the description at the expense of added computation of curve fitting and data smoothing. However, this smoothing and fitting process is performed *off-line* with a lenient time constraint. Hence, the trade-off is reasonable and beneficial.

After extracting the signatures, we build a hash table to index the space of invariant signatures. For a query protein structure, we compute its shape signatures using the same feature extraction procedure described above. Then, in the screening phase, we retrieve candidates of similar structures by using a voting mechanism based on the similarity of the hash keys. This allows efficient pruning of unlikely matching candidates, without expensive pairwise search of all proteins in the database.

For candidate proteins surviving the pruning process, we use a well-known dynamic programming algorithm (developed for sequence alignment [84]) to align pairwise the signatures of two proteins structures. The alignment result is a set of correspondences of structurally related residues. Those corresponding C_α atoms are superimposed and an RMSD (root mean square distance) value is computed for that subset. The length of the alignment and RMSD is used to compute a score [30] approximately reflecting the quality of the alignment. We also present the results of the alignment visually for further inspection. The main steps of our method can thus be summarized as follows:

For each protein in the database (an *off-line* process):

1. Calculate a spline fitting to best approximate the positions of the C_α atoms.
2. Compute, *for each residue,* curvature and torsion values at the C_α position along the spline. The secondary structure assignment of that residue is also recorded in the signature.

3. Compute a hash key based on the signature and store that in a hash table.

For a query protein (an *on-line* process):

1. Repeat steps 1 to 2 above and use the shape signature to screen the candidates from the hash table. Perform the following steps only for the candidates surviving the screening process.

2. For two proteins (a candidate database protein and the query), construct the normalized scoring matrix based on the distances between extracted features.

3. Run Smith-Waterman [84] local sequence alignment algorithm on the scoring matrix.

4. Superimpose the corresponding residues using a fast least-squares solution [5, 87].

5. Report results in an interactive visual form.

For each query, the first step in the on-line process is an efficient hashing based screening of the database of proteins, and the last four steps are for comparing two protein structures pairwise (the query and a candidate). A normalized scoring matrix is created on which the dynamic programming algorithm is run. A number of local regions with the highest alignment scores are chosen as candidates of structural similarity and passed to the final step of superimposition. For those highly similar regions, we superimpose C_α coordinates of the associated residues and check the RMSD of the alignment. We assign scores to them according to their lengths and

RMSD values and return the best scoring alignment as the best structural alignment. In the following sections we explain each step of our method in detail.

5.2.1 Spline approximation and error handling

The protein structure data are retrieved from the Protein Data Bank [8]. For each residue of the protein we obtain the 3D coordinates of its C_α atoms from the PDB file. As a result, each protein is represented by approximately equi-distant sampling points in 3D space. To construct a smoothing spline best approximating those points, we use the *Java AppLib* package (http://www.sscc.ru/matso/rozhenko/applib/), which is an approximation library for Java.

We use the quintic spline approximation, which is for 1-D curves. For a space curve, we use 3 independent smoothing splines parameterized with respect to the polygonal arc length t. The library package constructs the quintic smoothing spline, $\mathcal{C}(t)$, to given data, $\sigma(t_i)$, $i = 0, ..., n-1$, where n is the size of the C_α backbone, providing as small second derivative as possible (i.e., minimizing curvature). The method also ensures that the constructed spline does not deviate from the input data more than a given threshold by satisfying the following equation:

$$\sum_{i=0}^{n-1} w_i^{-1} \|\mathcal{C}(t_i) - \sigma(t_i)\|^2 \leq \varepsilon^2 \tag{5.1}$$

where w_i are positive weights, and ε is the maximum allowed deviation level. The larger a weight used for a residue, the greater the deviation is allowed. For our experiments we have used $w = 0.2$ for helices, $w = 0.4$ for strands, and $w = 2.0$ for turns. Also notice that, ε is a measure for the total deviation of the spline curve from

the data points. We have used a more intuitive measure, ε_0. It is an average error measure, not dependent on the length of the protein. We compute the ε in Eq. (5.1) with the following equation:

$$\varepsilon = \sqrt{\varepsilon_0^2 \cdot n} \tag{5.2}$$

Figure 5.1 shows an example of approximating 3D quintic spline for a small protein (1ei0:A), where individual spheres represent the C_α atoms and the dark colored 3D curve passing through them is the constructed smoothing spline. The average error estimate, ε_0, is 0.6 A^o. The curvature along the spline is minimized. In addition, as seen in the figure, we allow more smoothing of the data where there is a *turn* (the top part connecting two helices) and less smoothing where there is a *helix*.

5.2.2 Feature extraction

Curvature is defined as [22]:

$$\kappa = |\ddot{\mathcal{C}}| \tag{5.3}$$

And torsion is defined as:

$$\tau = \frac{1}{\kappa^2}[\dot{\mathcal{C}}\ \ddot{\mathcal{C}}\ \dddot{\mathcal{C}}] \tag{5.4}$$

where the square brackets have the special meaning of:

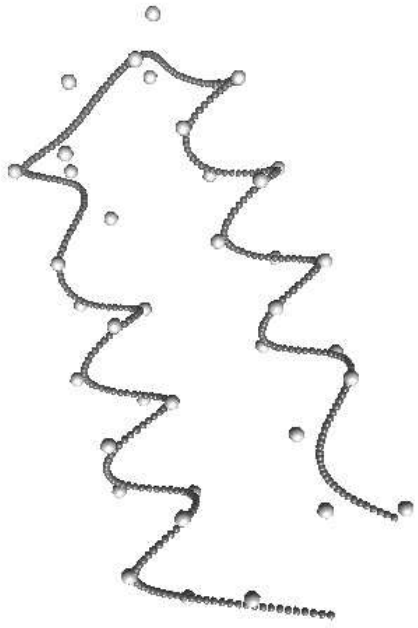

Figure 5.1: Spline approximation for C_α coordinates.

$$[\dot{\mathcal{C}}\ \ddot{\mathcal{C}}\ \dddot{\mathcal{C}}] = \begin{vmatrix} \dot{\mathcal{C}}_x & \dot{\mathcal{C}}_y & \dot{\mathcal{C}}_z \\ \ddot{\mathcal{C}}_x & \ddot{\mathcal{C}}_y & \ddot{\mathcal{C}}_z \\ \dddot{\mathcal{C}}_x & \dddot{\mathcal{C}}_y & \dddot{\mathcal{C}}_z \end{vmatrix} \tag{5.5}$$

In other words, the curvature of a point on the curve denotes how rapidly the curve pulls away from the tangent at that point, or how non-colinear a curve is. Similarly, the torsion of a point on the curve denotes how rapidly the curve pulls away from the osculating plane at that point, or how non-planar a curve is [22]. We compute the average curvature and torsion in a close neighborhood of a residue. Since computation of torsion involves the third derivative of the spline polynomial, we

have used quintic spline approximation, which guarantees the fourth order derivative continuity.

We also use the secondary structure assignment of a residue as a structural feature. Secondary structure assignment information is retrieved from the PDB web site [8]. PDB uses the DSSP method [48] to determine the secondary structure assignments of proteins. The signature value regarding the secondary structure assignment is one of the *helix*, *turn*, or *strand*. It should be noted that other biological properties of a residue, such as hydrophobicity, can also be used as part of the signature - if discrimination based on those traits are desirable.

5.2.3 Hashing for fast retrieval of candidates

After the feature extraction phase, we perform a quantization and normalization procedure on the curvature and torsion values. After that procedure each curvature and torsion value resides in the interval [0,255].

Each signature feature represents one dimension in our hash table. Therefore, we create a three dimensional hash table for curvature, torsion, and secondary structure type. To ensure a robust and reliable retrieval of candidates, the resolution of the hash table must be judicially chosen. The coarser the resolution, the smaller the size of the hash table gets. However, coarser resolutions reduce the discriminating power of the curvature-torsion descriptor and result in more false positive candidates surviving the screening process. On the other hand, finer resolutions increase the size of the hash table. The descriptor also becomes more susceptible to random error fluctuation, resulting in true positives being screened out. After some experimentation,

we have chosen the resolution of our hash table to be $64 \times 64 \times 3$.

For each signature triplet, (κ, τ, ss), we compute a hash key, which is simply $(\kappa/4, \tau/4, ss)$. By using that key as an index to the hash table, we store the signature triplet into the hash table along with its host protein chain identifier and its residue number. This process is executed *offline* for each protein in the database.

For a query protein, p, we extract its shape signatures as described in subsections 4.2.1 and 4.2.2. We perform similar quantization and normalization of the curvature and torsion values. For each residue of the query protein, we compute a hash key using its signature triplet, $(\kappa_p/4, \tau_p/4, ss_p)$. We retrieve the hash table entry indexed by that key. We accumulate a vote for each database protein stored in that entry. We repeat this process for each residue of the query protein. At the end, we assign a significance score, *a-score*, to each of the database proteins. We compute the *a-score* of a database protein by normalizing its accumulated votes by its length (number of residues). Proteins with higher *a-score*s are promising candidates that may have structural similarity to the query protein. We sort the proteins according to their *a-score*s and we select the top N queries. We use a cutoff value instead of a threshold, because *a-score* is not universal like a statistical significance score, i.e., it also depends on the length of the *query* protein. Finding a better scoring scheme is among our future work. For our experiments we have chosen N to be 200, which means we screen out approximately 90% of the database after the pruning phase.

By using the voting mechanism described above, we efficiently retrieve candidates of similar structures. Hence, we avoid exhaustive scan of the entire database.

5.2.4 Pairwise comparison

We perform pairwise comparison only for the candidates surviving the screening process. For two proteins under consideration, we first construct a normalized scoring matrix and then use a modified version of Smith-Waterman [84] dynamic programming algorithm on that distance matrix. The best scoring local alignment defines a set of correspondences of residues, which is then superimposed by a fast closed-form solution. The details of pairwise comparison are explained below.

Distance matrices

The distance matrices we compute should not be confused with the inter-atomic distance matrices of the DALI method [42]. The distance matrices we compute are the signature distance matrices between two proteins. The entry d_{ij}^{AB} of the distance matrix denotes the distance between the quantized and normalized signature values of the i^{th} residue of protein A and the j^{th} residue of protein B, and it is defined by the following equations:

$$d_{ij}^{AB} = \sqrt{(\kappa_i^A - \kappa_j^B)^2 + (\tau_i^A - \tau_j^B)^2} + s_{ij}^{AB} \tag{5.6}$$

$$s_{ij}^{AB} = \begin{cases} c, & \text{if} \quad SS(r_i^A) \neq SS(r_j^B) \\ -c, & \text{if} \quad SS(r_i^A) = SS(r_j^B). \end{cases} \tag{5.7}$$

where $SS(r_i^A)$ denotes the secondary structure assignment of the i^{th} residue of protein A. This measure is basically the Euclidian distance between the curvature and

torsion tuples, (κ_i^A, τ_i^A) and (κ_j^B, τ_j^B), regulated by the secondary structure assignment agreement. Agreement on secondary structure assignment decreases the distance by a constant c, and disagreement increases the distance by the same constant. We have used $c = 20$ in our experiments. Our choice of incorporating s_{ij}^{AB} into the signature as an offset instead of a multiplication factor is because the secondary structure assignments are not 100% correct, and may mislead the alignment if they are the dominant factor in the distance equation. Figure 5.2 shows an example distance matrix for the shape signature relationships between the proteins 1faz:A and 1ytf:D. Darker regions indicate higher similarity of signatures.

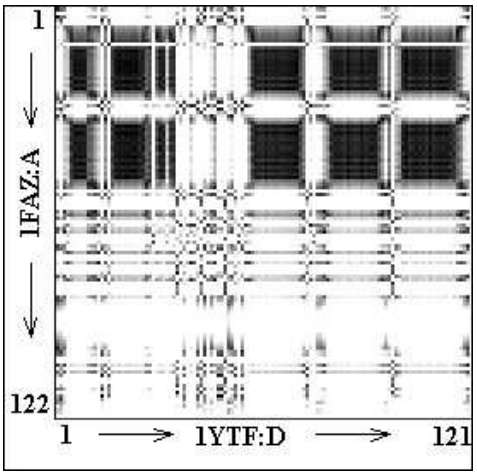

Figure 5.2: The distance matrix.

We convert those distance values to normalized score values in the interval [*low*, *high*] to be used by the local alignment in the dynamic programming phase. This is done by using the following equation:

$$score_{ij}^{AB} = low + \frac{-d_{ij}^{AB} + (256\sqrt{2} + c)}{256\sqrt{2} + 2c} \cdot (high - low) \tag{5.8}$$

We have chosen the low score to be -10.0 and the high score to be 20.0, because they define a range similar to that of the PAM matrix [20]. The score of the alignment makes sense this way by comparing it to the sequence alignment scores.

Local alignment by dynamic programming

The shape signatures can be thought of as protein sequence data with the alphabet, Σ=all possible triplets of **C**urvature, **T**orsion, and **S**econdary **S**tructure assignment. We define a similarity score between two signature values, Eq. (5.8), which is analogous to the entries of scoring matrices such as PAM [20] and BLOSUM [40] that define similarity scores between residue types. However, we do not compute a static scoring matrix, instead a distance matrix for each pair of proteins is created as explained in Section 4.2.4.

We then run the dynamic programming algorithm for sequence alignment by Smith and Waterman [84] using the dynamically computed and normalized scoring matrix. As in local sequence alignment we use an affine gap cost model, in which opening and extending gaps have different costs. For our experiments, we have used an opening gap penalty of 14 and an extending gap penalty of 10.

The complexity of alignment by using this method is $O(mn)$, where m and n are the numbers of residues in the compared proteins respectively. Figure 5.3 shows the best local alignment of 1faz:A and 1ytf:D on the distance matrix and the detection of a Helix-Turn-Helix (HTH) motif shared between those structures (3D result seen

in Figure 5.4). The diagonal light colored line shows the associated residues of two proteins.

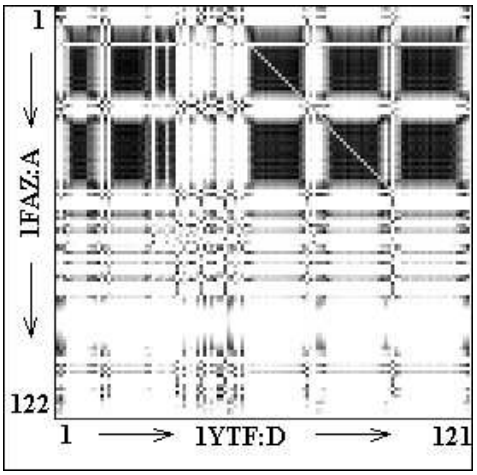

Figure 5.3: Local alignment with best score.

However, the best local alignment returned by the algorithm is not guaranteed to be the best structural alignment. Because of gaps in the alignment, the sub-structures represented by the alignment may actually have a high RMSD (e.g., those gaps may be regions of twists and turns affecting the overall alignment). Thus, we superimpose the query protein on the database protein and check the RMSD values of a number of best local alignments to obtain the best local alignment.

Superimposition

We use a fast least-squares solution to superimpose an ordered corresponding set of points in 3D space. Given a minimum of three pairs of point correspondences, the best rotation and translation, **R** and **T**, can be computed efficiently in $O(n)$ time,

where n is the number of corresponding points. A non-iterative least-squares solution based on the singular value decomposition (SVD) was suggested by Arun *et al.* [5] to find a closed-form solution. Umeyama [87] provided modifications to Arun *et al.* [5] to ensure that a correct rotation matrix, instead of a reflection, is computed when the data are noisy.

With the help of the superimposition, we compute the minimum RMSD values of the top local alignments. We assign a score to each alignment by using a similarity measure based on normalized RMSD [30] as defined by the following equation:

$$SCORE = \frac{length\ \ of\ \ alignment + 135}{225 * RMSD\ \ of\ \ alignment} \tag{5.9}$$

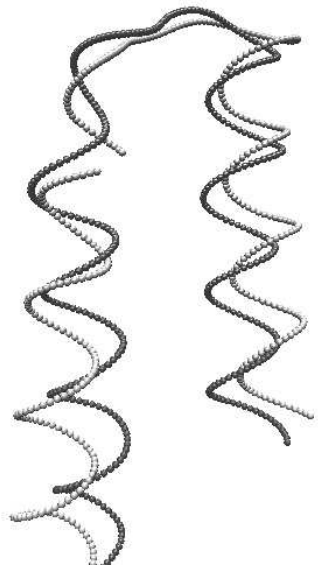

Figure 5.4: Superimposed local alignment result.

We return the best scoring alignment as the best structural alignment between the

compared protein structures. Figure 5.4 shows the superimposed result of the best local alignment found for 1faz:A and 1ytf:D. That alignment reveals a Helix-Turn-Helix motif shared between those protein structures. The Helix-Turn-Helix motif is usually found in DNA-binding proteins, and consists of a recognition helix and a stabilizing helix separated by a short loop.

5.3 Interactive Visualization of the Results

We use Java 3D graphics library[1] to visualize the results. The main advantage is that we do not have to generate visualization scripts and invoke external visualization tools like RASMOL in order to visualize the alignment results. Furthermore, our tool is platform independent and can be run within a web browser. Both the aligned 3D structures and the shape signature distance matrices are presented to the user. The result of the alignment is also shown to the user on the distance matrix. The user can select other suboptimal alignments and inspect their biological significance. Figure 5.5 shows the user interface with the shared motif between 2cro:_ and 2wrp:R.

5.4 Experiments

We have conducted two sets of experiments to evaluate the quality of the pairwise alignments produced by CTSS and to evaluate the sensitivity and running time performance of similarity search queries. We have chosen a widely used structural

[1] `http://java.sun.com/products/java-media/3D/`

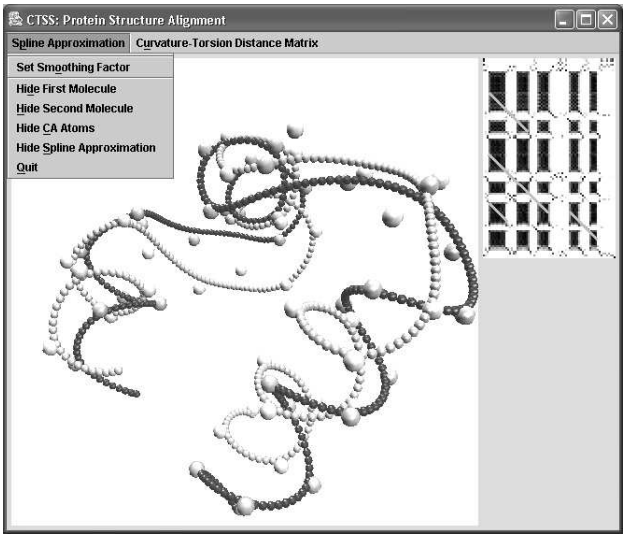

Figure 5.5: The user interface for CTSS.

alignment method, CE [80], to compare our results.

We present our experiment results in three subsections. In the first subsection, we present the results of pairwise alignment tests. In the second subsection, we assess the sensitivity of similarity search queries method by using the SCOP [64] structural classification and compare the run-times of the queries against CE's run-times. In the last subsection we present two example queries in detail for which we demonstrate visually the shared motifs we have discovered between the query proteins and a number of database proteins.

5.4.1 Evaluation of Pairwise Alignment Quality

There is no universally accepted measure to quantify the quality of a structural alignment [26]. However, we can use an approximate measure [30] that involves the

RMSD and the length of the aligned substructure. For this test we have selected the ASTRAL SCOP 1.63 database [13] that contains protein structures with less than 40% sequence identity to each other. Low sequence identity presents challenges to structure alignment algorithms, as it is not possible to use sequence similarity to predict structure similarity.

There are 5226 protein domains in the ASTRAL 40% database. 5187 of them are domains that span single chains. Out of these 5187 protein chains, 2952 chains contain single domains and we selected those as our test database. We selected single domain chains because this way we can uniquely assign a SCOP superfamily to a whole chain. We could not find structures (from Protein Data Bank) for 13 protein chains in this list, because they were either replaced by another protein chain in the PDB or they contained some errors that failed our structure parser. This leaves us with 2939 protein chains as our protein structure database. There are 4,314,453 ($2938 \times 2937/2$) possible pairwise alignments that we can perform using this database. However, the purpose of this test is to evaluate our method's ability to align well the structures with known similarity (i.e., from same SCOP superfamily). A similar test was performed by Gerstein and Levitt [30] to evaluate different methods on an earlier version of the SCOP database. Out of 4,314,453 pairs 16,300 of them are from the same SCOP family. Thus, we have conducted 16,300 pairwise alignments by both CTSS and CE.

For each pairwise alignment we have recorded the RMSD and length of the alignment. Figure 5.6 shows the RMSD/length plot for all the pairwise alignments conducted by CTSS and CE.

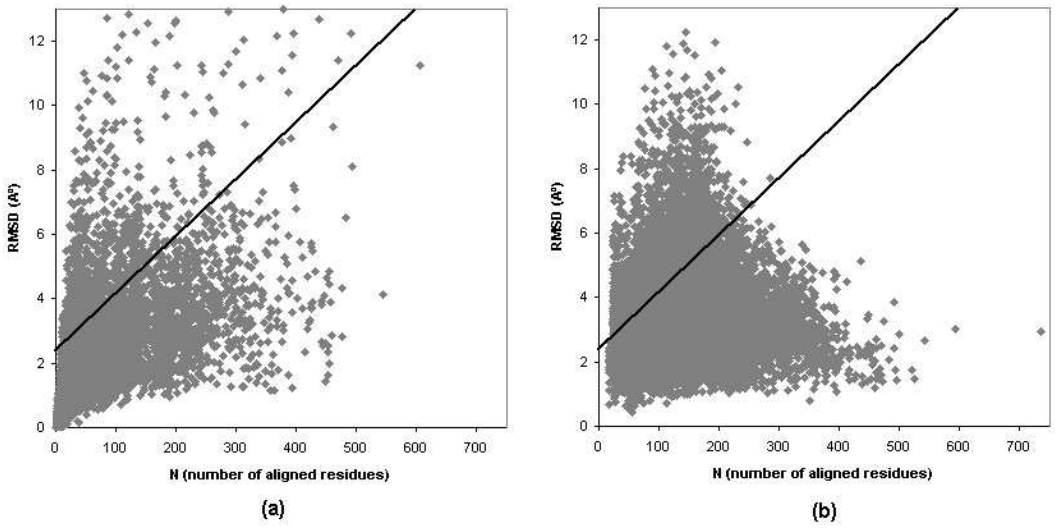

Figure 5.6: Pairwise alignment results, (a) CTSS (b) CE.

Method Name	*# of good*	*# of bad*	*avg. length of good*	*avg. RMSD of good*
CTSS	15103	1197	36.160	1.083
CE	12981	3319	155.063	3.099

Table 5.1: Pairwise alignment results

By using a demarcation line (the dark diagonal line), $RMS = 4(N+135)/225$ [30], it is possible to approximately separate the successful matches from unsuccessful matches. The points below the demarcation line are qualified as *good alignments* and the points above the line are *bad alignments*. Table 5.4.1 shows the number of *good alignments* and *bad alignments* as well as the average length and RMSD of *good alignments*.

It can be seen from the table that CTSS has more alignments that are quali-

fied as *good alignments*. This table also shows that the alignments found by CTSS is smaller than the ones found by CE. However, the average RMSD is also much smaller for those smaller alignments. In other words, we can say that on the average CTSS performs well for finding small shared substructures (motifs) accurately, i.e. with low RMSD.

5.4.2 Evaluation of Sensitivity and Runtime Performance of Similarity Search Queries

To evaluate the sensitivity of our method for performing similarity searches, we have used a benchmark similar to the one used by Fischer *et al.* [27] Our test database of 2939 protein chains comprises 867 different superfamily classes. 443 of these classes have more than one representative protein chain, i.e., 424 superfamilies have single members in the ASTRAL 40% database. We have selected a representative protein chain from each of the 443 superfamilies as the first alphabetic protein chain identifier (PDB ID) of that superfamily. This gives us 443 query (probe) protein chains to evaluate sensitivity. However, because of time constraints we have selected the first alphabetic 100 out of 443 protein chains. When we use CE to perform 100 queries on a database of 2939 proteins, it takes 293,900 pairwise alignments to finish (approximately 18 days on a single machine). Table 5.4.2 show the query proteins selected as benchmarks sorted by the number of member protein chains in their superfamilies.

The list of query proteins we have is a good representative of the SCOP database,

chain ID	*sf ID*	*# of sf members*	*chain ID*	*sf ID*	*# of sf members*
1a7j_	52540	69	1auia	56300	6
1a8q_	53474	45	1a2pa	53933	5
1a34a	49611	38	1a8ra	55620	5
1ajsa	53383	33	1af8_	47336	5
1alva	47473	33	1agre	48097	5
1aoy_	46785	32	1aoha	49384	5
1a7s_	50494	28	1atza	53300	5
1aq0a	51445	28	1auk_	53649	5
1a6m_	46458	26	1a59_	48256	4
1alu_	47266	26	1a6bb	57756	4
1a06_	56112	25	1agi_	54076	4
1aba_	52833	25	1ako_	56219	4
1ac6a	48726	25	1amx_	49401	4
1agg_	57059	24	1apxa	48113	4
1a8e_	53850	22	1atb_	57567	4
1a3k_	49899	21	12asa	55681	3
1aqb_	50814	21	1a32_	47060	3
1acw_	57095	20	1a3aa	55804	3
1avpa	54001	19	1a41_	56349	3
1avqa	52980	19	1a44_	49777	3
1akha	46689	18	1a4ya	52047	3
1awj_	50044	18	1a73a	54060	3
1aqca	50729	17	1a9na	52058	3
19hca	48695	15	1acf_	55770	3
1a3c_	53271	14	1ad1a	51717	3
1aac_	49503	14	1ad2_	56808	3
1ast_	55486	14	1ahsa	49818	3
1afra	47240	13	1am2_	51294	3
1a1w_	47986	12	1amua	56801	3
1a53_	51366	12	1ass_	52029	3
1agqa	57501	12	1at3a	50789	3
1a28a	48508	11	1avac	50386	3
1atx_	57392	11	1aw1a	51351	3
153l_	53955	10	1axn_	47874	3
1a0tp	56935	9	16pk_	53748	2
1a17_	48452	9	1a12a	50985	2
1afj_	55008	9	1a1x_	50904	2
1ak7_	55753	8	1a2za	53182	2
1aoea	53597	8	1a48_	56104	2
1apq_	57196	8	1a6f_	54211	2
1awcb	48403	8	1ab4_	56719	2
1a0aa	47459	7	1acz_	49452	2
1a4ma	51556	8	1aep_	47857	2
1a4sa	53720	7	1ah7_	48537	2
1aly_	49842	7	1aiw_	51055	2
1a0ca	51658	6	1ajj_	57424	2
1a6s_	47836	6	1ak4c	47943	2
1a7ta	56281	6	1akp_	49319	2
1a8i_	53756	6	1ap8_	55418	2
1adr_	47413	6	1auz_	52091	2

Table 5.2: List of query proteins

Class Name	*# of query proteins*
all alpha (α)	22
all beta (β)	21
alpha/beta (α/β)	24
alpha+beta ($\alpha + \beta$)	21
other (small, membrane, multi-domain)	12

Table 5.3: Class distribution of the query proteins

because the *class* distribution of the query set is roughly equal to the *class* distribution of the whole SCOP database. Table 5.4.2 shows the class distribution of the query list.

Given a query (probe) protein chain, the goal of similarity search is to retrieve proteins with similar structure from a database of protein structures. Another property of similarity search is that the retrieved results are sorted according to a similarity measure defined by the query method. A method with good sensitivity is expected to return members of its superfamily at the top ranks of the query results.

CE computes a statistical significance score, the *z-score*, for the structural alignment of two protein chains. We have used the *z-score*s of the results to sort the database proteins. CTSS uses a similarity measure defined by Equation 5.9. When executing these queries we ran CE for all the database proteins for pairwise comparison to the query protein. On the other hand, our method, CTSS, executes queries in two phases: the screening phase, and the pairwise comparison phase. It is important to note that in the screening phase we do not perform any detailed pairwise alignment. We use our hash index structure to find candidates of similarity by computing the *a-score*s for all the database proteins (refer to Section 4.2.3 for explanation of

Method Name	*at rank 1*	*at rank < 5*	*at rank < 10*	*overall score*
CTSS	55	73	82	0.646
CE	88	90	92	0.895

Table 5.4: The sensitivity assessment

the *a-score*).

For each of the 100 queries performed we registered the ranks at which a member from the same superfamily could be found. Table 5.4.2 shows the results of the similarity search queries. We counted the number of queries at which a member at rank 1, below rank 5, and below rank 10 could be found (excluding the query protein itself). In addition we computed the overall performance of the methods by the same equation defined by Fischer *et al.* [27], $\sum 1/r_i/|L|$, where r_i denotes the rank of the correct superfamily achieved by query protein i and $|L|$ is the number of query proteins in the benchmark: 100. For 92 of the queries, CE found a database protein that is in the same superfamily with the query protein in the first 10 results. CTSS accomplished this for 82 of the queries. The reason for CTSS performing worse than CE is that, CTSS is more focused on finding small structural motifs that may be detected in proteins from different superfamilies. Superfamily members usually share structural similarity in the larger scale and this can be detected better by CE. However, the performance of our method is still acceptable and as we show in the following paragraphs CTSS performed queries much faster than CE.

To evaluate the efficiency of CTSS, we have measured query execution times. The tests were run on a computer with dual AMD Athlon MP 1600+ processors with 2 GB of RAM, running Linux 2.4.19. We have used 1GB as the maximum size

of memory allocation pool for the Java Virtual Machine executing the query programs. The source code of CE is available publicly. To rule out disk I/O in timings, we have modified CE's reading of the protein structures. I.e., when performing an exhaustive search on a database of protein chains we load all proteins in the database into memory and do not consider the loading time when measuring the timings for CE.

Figure 5.7 shows the timing results. The timing for CTSS shows the total running time of pruning and pairwise comparison phases. Notice that the y-axis is plotted in log-scale. The average running time for CE is 11,877.42 seconds. The average running time for CTSS is 323.17 seconds. On the average our method is 36.8 times faster than the CE method. This is mainly because CE method need to do an exhaustive search by aligning all 2939 proteins to the query protein pairwise, whereas our method screens out 80% of the proteins very fast and performs pairwise queries for only 20% of the database that survive the screening process.

5.4.3 Detailed Example Query Results

Other than the alignment quality, and query sensitivity and performance tests, we have also performed more challenging queries for the protein chains 1faz:A and 1b16:A by using another data set that includes protein chains with less sequence identity. The data set consists of a representative set of proteins selected using the PDBSELECT method [41]. The PDBSELECT database is a subset of the structures in the PDB that does not contain homologue sequences, i.e., no two proteins have more than 25% sequence identity. As mentioned before, low sequence homologues

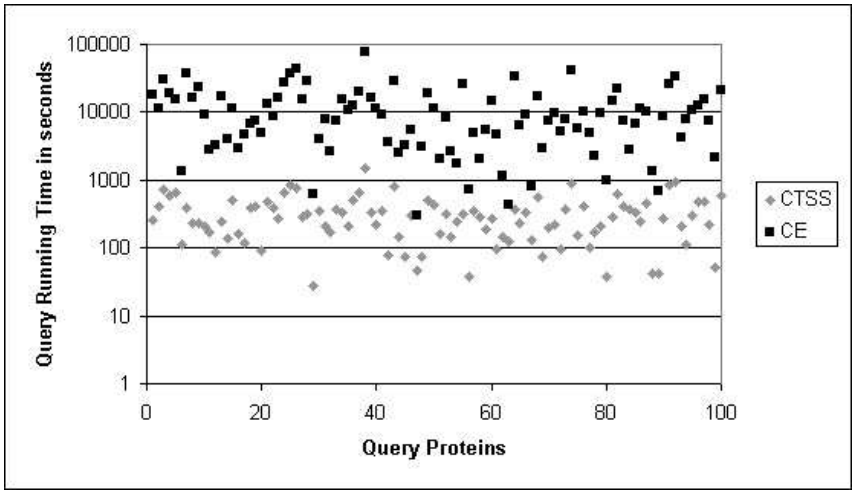

Figure 5.7: Timing results for the query dataset.

present challenges to structure alignment algorithms, as it is not possible to use sequence similarity to predict structure similarity. There are 1949 protein chains in that representative database (December 2002 version)[2]. Below, we present detailed results from those queries.

For the protein chains 1faz:A and 1ytf:D, we have found that they share a Helix-Turn-Helix motif, with length 42, and with RMSD 2.8 A^o. Those structures share only 1.9% sequence identity globally. The structural alignment program CE can find this alignment with length 52, but with much higher RMSD of 4.4 A^o. The result of that alignment is depicted graphically in Figure 5.4. This figure and the subsequent alignment figures show only the aligned parts of the protein chains. The unaligned (non-similar) parts are not shown for a more clear presentation of the aligned parts.

We have also discovered some motifs not detected by other alignment tools, such

[2] `http://homepages.fh-giessen.de/~hg12640/pdbselect/`

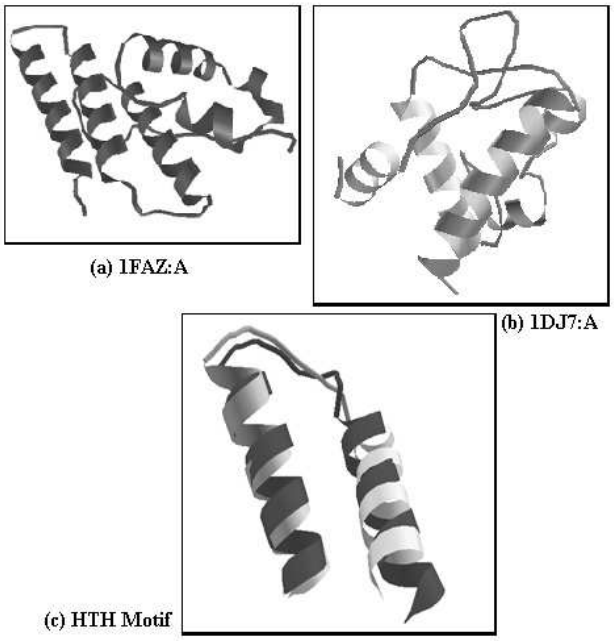

Figure 5.8: Helix-turn-helix match between 1faz:A and 1dj7:A.

as CE [80] or DALI [42]. For example, we have found the Helix-Turn-Helix motif between 1faz:A and 1dj7:A, with length 38 and RMSD 3.68 A^o, and with 2 gaps in 1dj7:A. Figure 5.8 shows the two proteins separately and the shared motif between them.

Another motif that was not detected by others and discovered by our program was between 1b16:A and 1h05:A. The length of that Helix-Strand-Helix motif is 35 with RMSD 3.26 A^o. Figure 5.9 shows that motif.

We have conducted a comparison between 1b16:A and 1gci:_. They share a Helix-Strand-Helix-Strand (HEHE) motif and the length of the alignment was 46, with RMSD 3.34 A^o. Those protein chains have 8.5% sequence identity. That shared

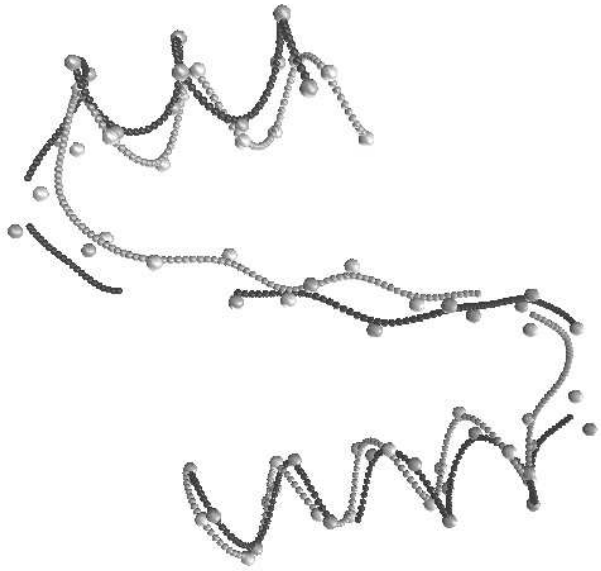

Figure 5.9: Shared motif between 1b16:A and 1h05:A.

motif can be seen in Figure 5.10.

Our program does not only find small motifs between protein structures. In our test cases we have also found longer structural alignments. 1b16:A and 1oaa:_ alignment has length 209 with RMSD 4.6 A^o.

Figure 5.11 shows the Strand-Helix-Strand motif discovered between 1b16:A and 1qp8:A. We have found a substructure match of length 35, with RMSD 1.58 A^o, and with two gaps of length one. Those proteins share 8.1% sequence identity.

Finally, we have also conducted a detailed pairwise comparison test between the protein structures 2cro:_ and 2wrp:R. Those structures were previously compared by Pennec and Ayache [70]. We have found a longer motif between those structures compared to what they reported and moreover our program can detect that shared

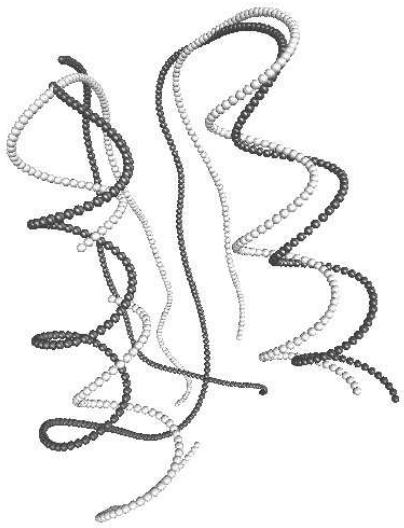

Figure 5.10: The helix-strand-helix-strand motif between 1b16:A and 1gci:_.

substructure in just a fraction of a second (compared to 18 seconds reported in Pennec and Ayache [70]). The alignment result can be seen in Figure 5.5.

5.5 Discussion

In this chapter, we have presented a new method for protein structure alignment. Our method comprises a novel technique for extracting compact and localized shape signatures for protein structures, an indexing component based on hashing to avoid an exhaustive scan of the entire structure database, and a pairwise alignment method for accurately aligning shape signatures even in the presence of small gaps.

The novelty of the proposed technique lies in the methods of feature design, extraction, and smoothing. Together, these methods ensure the success of the ensuing

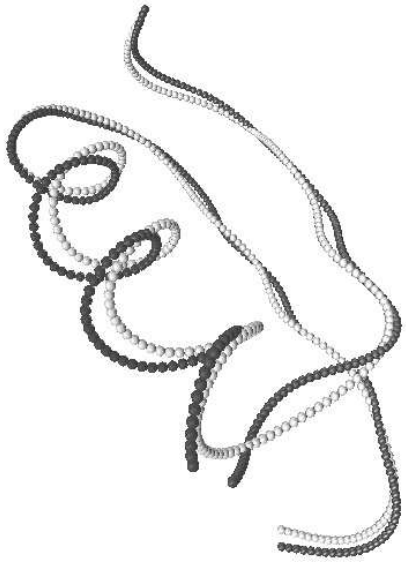

Figure 5.11: The strand-helix-strand motif between 1b16:A and 1qp8:A.

phases of protein structure screening and pairwise alignment. Extracting localized features and embedding both geometric and biological information in the signature are the major difference compared to other structural alignment methods. We have also employed an efficient screening phase based on hashing followed by an accurate pairwise alignment phase that can handle gapped alignments efficiently. Our experiments showed that our technique is able to execute protein structure similarity searches efficiently and discover biologically meaningful motifs shared between protein structures that were overlooked by other methods.

However, one question that comes to mind is how descriptive these localized signatures $(O(n))$ are in representing the structure of a protein? It is a fair question, especially when one considers that other existing algorithms generate signatures in

the order of $O(n^2)$ or $O(n^3)$ to capture the structure of a protein of length n. Here, we have the support of the theory of differential geometry, which states that space curves generating the same curvature and torsion values are isomorphic. It is true that the existence of gaps along the alignment of curves fails that theory. However, as the computation of curvature and torsion is a localized process, local isomorphism can still be detected based on such localized signatures. CTSS does not address the problem of finding alignments consisting of multiple structural fragments that are separated by large gaps. Obviously, that problem, which is tried to be solved by other methods, requires more computation (NP-hard). However, we present an approximation to that problem and this helps us identify small structural motifs very quickly and accurately. Our initial experiment results proved that our technique is a promising one. We have been able to find shared motifs between protein structures efficiently.

We are currently investigating indexing methods other than hashing for faster and more accurate screening of candidates of structural similarity. The *a-score*, defined to assess the significance of a match (in the pruning phase), has room for improvement. We will work on finding methods to incorporate the *statistical* significance of the accumulated votes to the *a-score*.

The analogy between sequence and structure comparison introduced by our method promises other benefits. Existing sequence comparison algorithms like BLAST [1], which can conduct very efficient sequence similarity searches, can be tailored to conduct structure similarity searches (or multiple structural alignments) with the use of our localized shape signatures that is a one to one mapping to the sequence (i.e.,

each residue has one localized signature value).

Chapter 6

Simultaneous Protein Sequence and Structure Similarity Search

The most profitable research in bioinformatics often results from integrating multiple sources of data [29]. For instance, the 3D coordinates of a protein are more useful if combined with data about the protein's function, occurrence in different genomes, and interactions with other molecules. In this way, individual pieces of information are put in context with respect to other data.

In this chapter, we consider the problem of similarity searches on protein databases based on both sequence and structure information simultaneously [10]. As the number of known protein structures increases conducting similarity searches in a reasonable amount of time becomes more important. Current techniques used for similarity searches usually involves a sequence similarity query followed by a structure similarity query for the top results of the sequence query results. However, we claim that if a simultaneous query that incorporates both the sequence and structure information is more efficient and less error prone than a sequence query followed by a

structure query.

Our technique presented in this chapter extracts feature vectors from both the sequence and structure components of the proteins. These feature vectors are then combined and indexed using a novel multi-dimensional index structure. For a given query, we employ this index structure to find candidate matches from the database. We develop a new method for computing the statistical significance of these candidates. The candidates with high significance are then aligned to the query protein using the Smith-Waterman technique to find the optimal alignment. The experimental results show that our method can classify up to 97 % of the superfamilies and up to 100 % of the classes correctly according to the SCOP classification. Our method is up to 37 times faster than CTSS (that was detailed in Chapter 5), combined with Smith-Waterman technique for sequences.

6.1 Introduction

The industrialization of molecular biology research has resulted in an explosion of bioinformatics data (DNA and protein sequences, protein structures, gene expression data and genome pathways). Each of these data present a different type of information about the functions of the genes and the interactions between them. Most of the earlier work focuses on only one type of data since each type of data has a different representation and the means of similarity varies for each data type. Combined learning from multiple types of data will help biologists achieve more precise results for several reasons: a) The probability of having false positive results due to errors

in data generation decreases since it is less likely for the same error to appear in all the datasets. b) More than one aspect of the biological objects can be captured simultaneously.

As an example, for proteins, functionally similar homologs can be found by the sequence similarity of the proteins. On the other hand, distant homologs that have similar functionality can be revealed by the 3-D structure similarity of the proteins. A combined search based on both sequence and structure can find the entire functionally related set of proteins for a given query protein. Such a search would reveal more precise information about the function and the classification of a new protein. Moreover, new insights on the relationship between sequence and structure can be gained with the help of combined searches.

6.1.1 Problem definition

In this chapter, we consider the problem of joint similarity searches on protein sequences and structures. A protein is represented as an ordered list of amino acids, where each amino acid has a sequence and a structure component (the terms amino acid and residue are used interchangeably). The sequence component of an amino acid is its residue name indicated by a one letter code from a 20 letter alphabet. The structure component consists of the Secondary Structure Element (SSE) type of that residue (α-helix, β-sheet, or turn), and a 3-D vector which shows the position of its carbon-alpha (C_α) atom.

A query is specified by a four-tuple $<Q, \epsilon_q, \epsilon_t, \tau>$, where Q is a query protein, $\epsilon_q \in [0, 1]$ and $\epsilon_t \in [0, 1]$ are the distance thresholds for se*q*uence and s*t*ructure

components, and τ is the boolean value regarding the use of SSE information. A sample query may be as follows:

"Find the protein chains in PDB that have sub-patterns whose sequence differs by at most 20 % and structure differs by at most 5 % from that of 1f53-A, and the matching residues are of the same type of SSEs as that of 1f53-A."

In this example, the query tuple is $<Q$ = 1f53-A, $\epsilon_q = 0.2, \epsilon_t = 0.05, \tau =$ *true*$>$. The values for ϵ_q and ϵ_t, $0 \leq \epsilon_q, \epsilon_t \leq 1$, denote the importance of the sequence and structure information to the user respectively. Smaller values for these parameters results in closer matches. Similarly, τ = *true* means that the types of the SSEs should match and τ = *false* ignores the SSE types.

6.1.2 Related work

It has been one of the most important goals in molecular biology to elucidate the relationship among sequence, structure, and function of proteins [90, 39, 76]. A handful of algorithms and tools have been developed to analyze sequence and structure similarities between the protein molecules. These methods are usually focused on either the sequence (Smith-Waterman (SW) [84], BLASTP [1, 34], PSI-BLAST [3]) or the structure information (VAST [60], DALI [42], CE [80], PSI [15], CTSS [17]) for finding similarities between different proteins.

On the other hand, a few tools have been developed for providing integrated environments for analyzing the sequence and structure information together. Protein Analyst [75], 3DinSight [4], and the integrated tools by Huang *et al.* [46] are among those tools. They provide a combination of separate (but cooperating) programs for

integration of sequence and structure analysis under a single working environment. The components of these systems are usually run one after another, with one's results being the input to the other. Unlike these tools, JOY [63] executes a single alignment program (sequence or structure), and it also provides additional information (e.g., structural features such as SSE type) on the resulting alignments in terms of annotations.

Although these tools provide integration of multiple types of data, they perform search on only one type of data at a time. We believe that integration of multiple data sources at indexing and search level would provide more precise and efficient tools.

6.1.3 An overview of our method

We extract feature vectors on sequence and structure components of proteins by sliding a window on each protein in the database. Each feature vector maps to a point in a multi-dimensional space. This multi-dimensional space consists of orthogonal dimensions for sequence and structure. Later, we partition the space with the help of a grid and index these points using Minimum Bounding Rectangles (MBRs).

Given a query, our search method runs in three phases:

Phase 1 (index search): Feature vectors (i.e., points) are extracted from the query protein. For each of these query points, all the database points that are within ϵ_q and ϵ_t distance along the sequence and the structure dimensions are found using the index structure. Each such point casts a vote for the protein to which it belongs as in geometric hashing [89].

Phase 2 (statistical significance): For each database protein, a statistical significance value is computed based on the votes it obtained in Phase 1 and its length.

Phase 3 (post-processing): The top c proteins of highest significance are selected, where c is a predefined cutoff. The optimal pairwise alignment of these c proteins to the query protein are then computed using the SW technique. Finally, the C_α atom of the matching residues are super-positioned using the least-squares method by Arun *et al.* [5] to find the optimal RMSD (Root Mean Square Distance).

We name our method as *ProGreSS* (*Pro*tein *Gre*p by *S*equence and *S*tructure) since it enables queries based on sequence and structure simultaneously.

The contributions of this work can be summarized as follows:

1. A new query model that incorporates both sequence and structure is defined.

2. A new method that maps protein sequences into a multi-dimensional space, using a sliding window, based on a given score matrix is developed.

3. A novel index structure that stores synopsis for sequence and structure of proteins simultaneously is proposed.

4. A new method that computes the statistical significance of the matches in the index structure is developed.

6.2 Feature vectors and index construction

In this section, we develop new methods to extract features for protein structures and sequences. Feature vectors for structures are computed as the curvature and torsion values of the residues in a sliding window. Curvature and torsion values provide

a necessary and sufficient condition for the isomorphism of two space curves [17]. Feature vectors for sequences are computed using a sliding window and a score matrix that defines the similarity between all the amino acids. We also propose a novel index structure to provide efficient access to these features.

6.2.1 Feature vectors for structure

We slide a window of a pre-specified size, w, on the proteins (i.e., each positioning of the window contains w consecutive residues). We will discuss the choice of w later. Figure 6.1(a) depicts two positionings of the window. For a given window, the curvature and torsion values for each residue in that window is computed. The resulting vector contains $2w$ values since two values are stored per residue in the window. This vector maps to a point in a $2w$-dimensional space. Having a large number of dimensions increases the cost for computing the similarity [9] and the cost for storing the vectors. Therefore, we reduce the number of dimensions to a smaller number, d_t, using the Haar wavelet transformation, at the cost of reduced precision. We use d_t = 2 in our experiments. The transformed vector is normalized to $[0,1]^{d_t}$ space. Along with each feature vector, we also store the SSE types of the residues.

As w increases, the feature vector contains information about the correlation between larger number of residues. Thus the similarity between two feature vectors implies longer matches. On the other hand, very large values for w may cause false dismissals since shorter matches may be discarded due to their neighboring residues. We set w = 3 for our experiments.

6.2.2 Feature vectors for sequence

The similarity between two amino acids of protein sequences is usually defined using score matrices (e.g., PAM and BLOSUM). A score matrix consists of 20 rows and columns; one for each amino acid. The entries of a score matrix denote the score for aligning a pair of residues. If two amino acids are similar (e.g., having similar chemical properties or being close in mutation cycle), then the score for that pair is large, otherwise it is small.

Given a score matrix M, we call each row of M the *score vector* of the amino acid corresponding to that row. Thus, each entry of this vector shows the similarity of that amino acid to one of the 20 possible amino acids. We define the distance between two amino acids as the Euclidean distance between their score vectors. This is justified, because if the score of aligning two amino acids x and y is high in a score matrix, then they are similar. Therefore, if x is similar (or dissimilar) to another amino acid z, then y is also similar (or dissimilar) to z.

Similar to protein structures, we extract feature vectors for protein sequences by sliding a window of length w (see Figure 6.1(b) for w = 3.). Each positioning of the window contains w amino acids. We append the score vectors of these amino acids in the same order as they appear in the window to obtain a vector of size $20w$. This vector maps to a point in $20w$-dimensional space. Since the number of dimensions is too large for efficient indexing even for small values of w, we reduce the number of dimensions to a smaller number, d_q, using Haar wavelets. Similar to the structure component, we recommend d_q = 2 for optimal quality/time trade-off. The resulting vector is then normalized to $[0,1]^{d_q}$ space. We again choose w = 3.

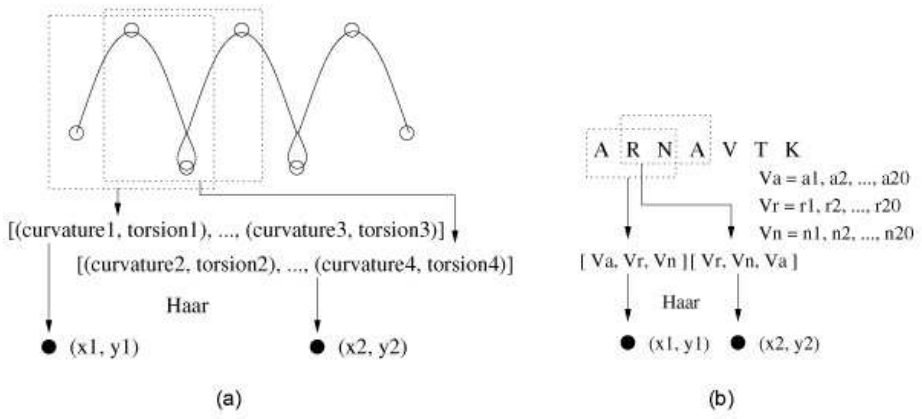

Figure 6.1: Feature vectors for (a) protein structure, and (b) protein sequence.

6.2.3 Indexing feature vectors

So far we have discussed how to extract feature vectors for structure and sequence components of the proteins separately. In this section, we will discuss how to build an index structure on these feature vectors.

In order to search the protein database based on both sequence and structure, we need to combine the feature vectors for these two components. Since the same window size is used for both the components, every positioning of the window produces one d_t-dimensional feature vector for its structure component and one d_q-dimensional feature vector for its sequence component. We append these two vectors to obtain a single (d_t+d_q)-dimensional vector. The resulting vector is called the *combined feature vector*. Since the entries of each of the feature vectors are normalized to the $[0,1]$ interval, the combined feature vector resides in a $[0,1]^{d_t+d_q}$ space

```
/* Let D be a dataset that contains proteins,
       w be the window size,
       V be the volume cutoff. */
Procedure CreateIndex(D, w, V)
for each protein x ∈ D
    for each positioning of window of length w
        p := combined frequency vector for current window;
        C := cell that contains p;
        if C = ∅ then
          B.Lower := p;
          B.Higher := p;
          Insert B into C;
        else
          B := argmin_{B∈C}{volume(B ∪ p)};
          if volume(B ∪ p) ≤ V then
            B := B ∪ p;
          else
            B.Lower := p;
            B.Higher := p;
            Insert B into C;
          endif
        endif
    endfor
endfor
```

Figure 6.2: Algorithm for building the index structure.

(called the *search space*).

The index structure is built by first partitioning the search space into η equal pieces along each dimension. The resulting grid contains $\eta^{d_t+d_q}$ cells of length $1/\eta$ along each dimension. (We will discuss the choice of η in Section 6.3.1.) Once the space is partitioned, a window of length w is slid on each protein in the database. For each positioning of the window, the combined feature vector is computed. Each combined feature vector maps to a point p in one of the cells of the grid. For each such point, we check whether that cell is empty. If it is empty, we construct an MBR

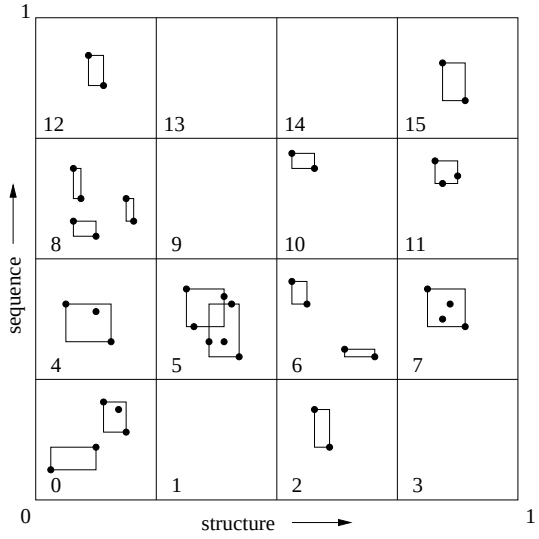

Figure 6.3: A layout of the MBRs and data points on the search space for η = 4 in 2-D.

that contains only p. Otherwise, we find the MBR B in that cell whose volume becomes the smallest after extending it to contain p. If the volume of B, after its expansion, is less than a pre-computed volume threshold, V, then we extend B and insert p into B, otherwise we create a new MBR that covers only p. Figure 6.2 presents the algorithm that constructs the index structure. Figure 6.3 depicts a layout of the search space and the MBRs built on the data points for η = 4 in 2-D. In this example, $d_t = d_q$ = 1.

As V decreases, the MBRs in the index structure becomes more compact. On the other hand, space cost and running time of the index structure increases since the number of MBRs increases. In our experiments, we observed the best performance for $V = (1/2\eta)^{d_t+d_q}$.

6.3 Query method

Given a query $<Q, \epsilon_q, \epsilon_t, \tau>$, where Q is a query protein, $\epsilon_q \in [0, 1]$ and $\epsilon_t \in [0, 1]$ are the distance thresholds for sequence and structure, and τ is the boolean value regarding the use of SSE information, our search algorithm runs in three phases: 1) index search, 2) statistical significance computation, and 3) post-processing. In this section, we will discuss each of these phases. We will assume that the index structure is built using a user specified score matrix for sequence (e.g., PAM or BLOSUM), and w for the window size.

6.3.1 Index search

Each residue of the query protein Q consists of a sequence component and a structure component. We extract combined feature vectors from Q by sliding a window of length w on it. Each of these combined feature vectors defines a query point in the search space. Figure 6.4 shows a sample query point in a 2-D search space, where the horizontal axis is the structure dimension and the vertical axis is the sequence dimension. In this figure, the search space is split into 16 cells numbered from 0 to 15. The query point falls into cell 10. We want to find the database points that are within an ϵ_t distance along the structure dimensions and ϵ_q distance along the sequence dimensions from the query point. In Figure 6.4, we are interested in the points in the shaded region. Note that if τ = *true*, then we only consider the database points that have the same SSE type as the query point.

For each query point, we construct a query box by extending it by ϵ_t and by ϵ_q

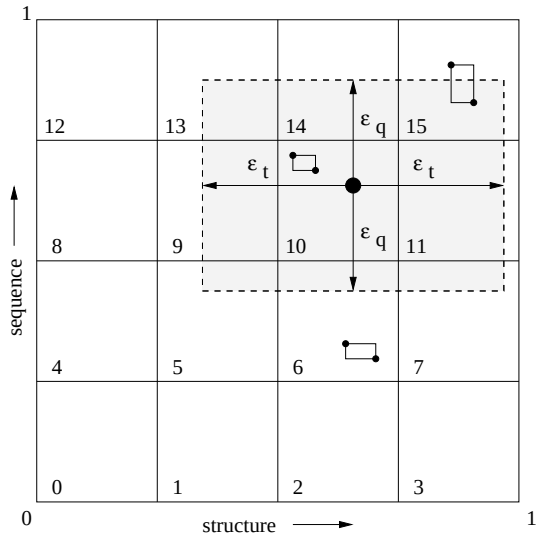

Figure 6.4: A sample query point and its query box for $\eta = 4$ in 2-D.

in both directions along the structure and the sequence dimensions respectively (see Figure 6.4). Next, we find the cells in the search space that overlap the query box. If a cell does not overlap the given query box, then it is guaranteed that it does not contain any database points that are in the query box. A cell can overlap a query box in two ways: 1) it is contained in the query box (e.g., cell 10 in Figure 6.4), or 2) it partially overlaps the query box (e.g., cells 5, 6, 7, 9, 11, 13, 14, and 15 in Figure 6.4).

1) If a cell is contained in the query box, all the points in that cell are guaranteed to overlap the query box. Therefore, we increment the vote to the database proteins that contains a data point in that cell for each such data point (if τ = *true*, then the vote is added only for the points that have the same SSE type as the query point).

2) If a cell partially overlaps the query box, then we check all the MBRs in that cell. If an MBR is contained in the query box (e.g., the MBR in cell 10), each point in that MBR contributes a vote. If an MBR partially overlaps the query box (e.g., the MBR in cell 15), then we find the points in that MBR that are in the query box to find the votes. If an MBR does not overlap the query box (e.g., the MBR in cell 6), we ignore all the points in that MBR. This method is more precise than geometric hashing [89], because for a given query point it inspects the neighboring cells as well as the cell into which that query point falls.

The number of partitions η in the search space affects the run time of the index search. As η decreases, each cell contains more MBRs. Therefore, if a query box partially overlaps a cell, then more MBRs need to be tested for intersection with the query box, thus increasing the search time. For example, if η = 1, then there is only one cell which is equal to the search space. So, all MBRs are tested for intersection for all the query boxes. On the other hand, having too many partitions have two disadvantages: 1) most of the cells will be sparse or empty incurring space cost. 2) the volume of the boxes will be very small since each cell will get smaller. This increases the total number of MBRs, and hence the number of MBRs for intersection test. In our experiments we recommend η = 10 for optimal results.

6.3.2 Statistical significance computation

Once the index structure is searched, we obtain a number of votes for each protein in the database. The total number of votes for a protein x shows the number of query points that are close to x's points. Assume that proteins x and y have 80 and 200

points in the index structure respectively. For a given query, assume that both x and y have 100 votes. We would like to determine which of these proteins is more likely to be a better match to the query. Intuitively, the answer is x since it has smaller number of points, and thus it is less likely for x to have the same number of votes as y. We define the *p-value* of a match as the *unexpectedness* of that result. Smaller p-values imply better matches.

Definition 1 *Given a protein x with n points in the index structure and v votes for a given query, the* p-value *of x for that query is defined as the probability of having at least v votes for a randomly generated protein with n points in the search space.*

Next, we discuss the computation of p-values. Consider a protein in the database that is represented in the search space using n points (n = length of protein – window size + 1). Let the protein receive v votes for a given query. Let X be a random variable representing the number of query boxes that overlap with a randomly selected point in the search space. Let μ_X and σ_X^2 be the mean and the variance of X. The total number of query boxes that overlap with n randomly selected points can be computed as $X_n = X + X + \cdots + X$ (exactly n Xs). Since Xs are independent and identically distributed random variables, using Central Limit Theorem, one can show that X_n is normally distributed with mean $\mu_{X_n} = n \cdot \mu_X$ and variance $\sigma_{X_n}^2 = n \cdot \sigma_X^2$. Thus, if μ_X and σ_X^2 are known, one can compute the distribution of X_n using a normal distribution. Since the protein has v votes, its p-value can be computed as $P(X_n \geq v)$.

The computation of p-values requires the values of μ_X and σ_X^2. The distribution of X depends on the distribution of query points, and the distance thresholds ϵ_q and

ϵ_t. We compute the values of μ_X and σ_X^2 by generating a large number of random points in the search space and counting the number of query boxes that it overlaps. In our experiments, we generate 10,000 random points for this estimation.

6.3.3 Post-processing

After the statistical significances of all the proteins are computed, top c proteins with the highest significance are selected as candidates for post-processing, where c is a predefined cutoff. The purpose of post-processing is to find the optimal alignment between the query protein and the most promising proteins. Let q be the query protein. For every protein x in the candidate set, post-processing runs in two steps:

Step 1: We build a $|x| \times |q|$ score matrix, M_{str}, for structure component, where $|x|$ and $|q|$ are the number of residues in x and q as follows: For each residue in x and q, we construct a 2-D vector as its curvature and torsion. Each entry of M_{str} is then computed as the negative of the Euclidean distance between the <curvature, torsion>-vector of the corresponding residues. This strategy is also used by CTSS as described in Chapter 5. For the sequence component, we create another $|x| \times |q|$ score matrix, M_{seq}, such that $\forall i, j$ the entry $M_{\text{seq}}[i, j]$ is equal to the score of aligning the i^{th} letter of x with the j^{th} letter of q in the underlying score matrix (e.g., BLOSUM62). Later, a combined score matrix $M_{\text{com}} = (1 - \epsilon_t) \cdot M_{\text{str}} + (1 - \epsilon_q) \cdot M_{\text{seq}}$ is computed. Here, the weights $(1 - \epsilon_t)$ and $(1 - \epsilon_q)$ represent the importance that the user gives to each of the components. The optimal alignment between x and q is then found by running the Smith-Waterman dynamic programming using M_{com}.

Step 2: The alignment obtained in Step 1 defines a one-to-one mapping between

a subset of residues of x and q, and is optimal with respect to M_{com}. Finally, we find the 3-D rotation and translation of x that gives the minimum RMSD to q by using a least-squares fitting method [5].

6.4 Experimental evaluation

We used single domain chains in our experiments. We downloaded all the protein chains in PDB (`http://www.rcsb.org/pdb`) that contain only one domain according to VAST and SCOP [64] classifications. We only considered proteins that are members of one of the following SCOP classes: all α, all β, α+β and α/β. We identified the superfamilies (according to SCOP classification) that have at least 10 representatives in this dataset. There are 181 such superfamilies. We created a database D of size 1810 proteins by including 10 proteins from each of these superfamilies. We formed a query set, D_Q, by choosing a random chain from each of the 181 superfamilies in D. D_Q is large enough to sample D since it contains one protein from each superfamily. We ran a number of experiments on these sets to test the quality and the performance of ProGreSS. The tests were run on a computer with two AMD Athlon MP 1600+ processors with 2 GB of RAM, running Linux 2.4.19.

In the rest of this section, we use w for the window size, c for the cutoff, ϵ_t and ϵ_q for the structure and sequence distance thresholds, τ for the SSE type match choice, and η for the number of partitions. We employ the BLOSUM62 score matrix for sequences in all of our experiments. The number of dimensions d_q and d_t for sequence and structures are both set to 2.

6.4.1 Quality test

Our first experiment set inspects the effect of various indexing and search parameters on the quality of our index search results. We classify a given query protein into one of the superfamilies and classes using the c best seeds as follows. The logarithms of the p-values of the matches in top c results in each superfamily are accumulated. The query protein is classified into the superfamily that has the largest magnitude of this sum. We use the same technique to classify the query protein to one of the four SCOP classes: all α, all β, α+β and α/β. We do not report the classification results for folds since they were similar to that for superfamilies. Since the queries are selected from the database, in order to be fair, we do not take into account the query protein itself if it is among the top c results. We will only report the results for τ = *true*, since it usually produced slightly better results than τ = *false*.

Figure 6.5 shows the percentage of query proteins correctly classified to classes (CL) and superfamilies (SF) for different values of c, where $\epsilon_t = \epsilon_q$ = 0.01 and 0.02, and w = 3. In all these experiments, we obtained the best results for c = 2 and 3. We achieved up to 96 % and 94 % correct classification for classes and superfamilies respectively. As c increases, our method starts retrieving proteins from other classes and superfamilies. We set c = 3 for the rest of the experiments.

Figure 6.6 plots the percentage of correctly classified proteins for varying distance thresholds when $\epsilon_t = \epsilon_q$ and w = 3. The purpose of this experiment is to understand what a good distance threshold should be when sequence and structure have equal importance. The graph shows that the accuracy of ProGreSS increases when distance threshold increases from 0.005 to 0.01. At $\epsilon_t = \epsilon_q$ = 0.01, ProGreSS

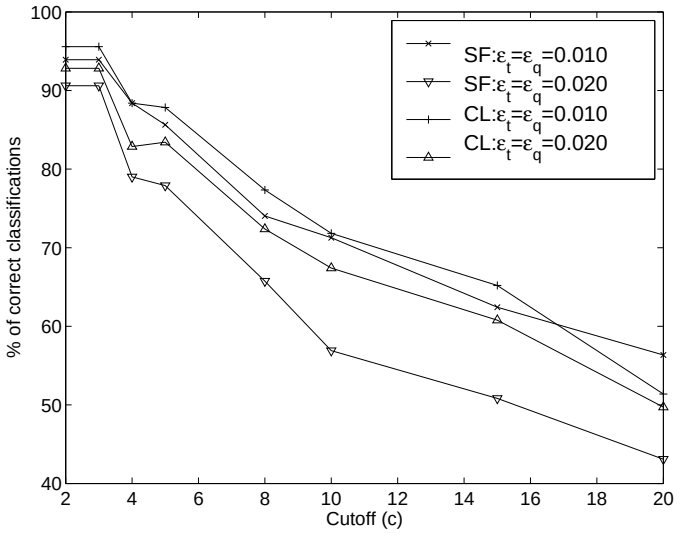

Figure 6.5: Percentage of query proteins correctly classified for different values of c.

achieves 96 % and 94 % correct classification for classes and superfamilies. As the distance threshold increases, the accuracy of ProGreSS drops. This is because it starts retrieving distant proteins.

Figure 6.7 shows the percentage of correctly classified superfamilies for different values of ϵ_t when ϵ_q is fixed and for different values of ϵ_q when ϵ_t is fixed, for w = 3. The purpose of this experiment is to see the effect of distance threshold for each of the structure and sequence components separately. When ϵ_q is fixed, as ϵ_t decreases, the classification quality of ProGreSS increases. This implies that our method can find better results when the distance threshold is small. The highest accuracy obtained is 62 %. For ϵ_q = 1.0 (i.e., when the sequence component is ignored), ProGreSS performs the worst. This is an important result since it shows that

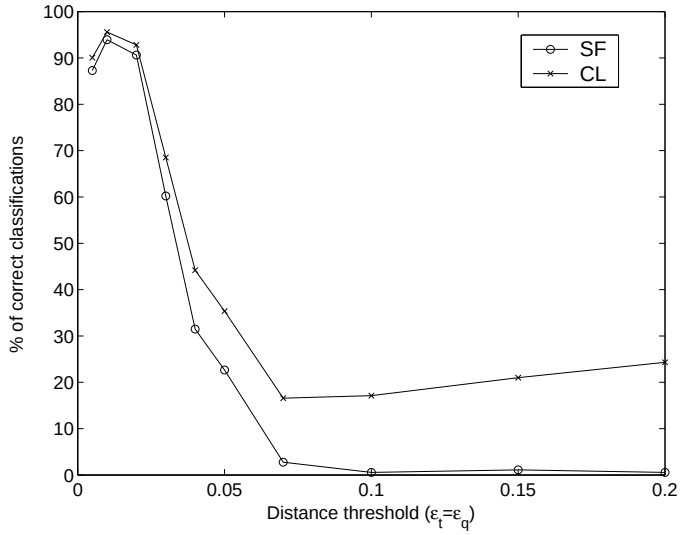

Figure 6.6: Percentage of query proteins correctly classified for different values of distance threshold when $\epsilon_t = \epsilon_q$.

searches based on structure alone would incur more false positives than the searches based on both sequence and structure. When ϵ_t is fixed, as ϵ_q decreases, ProGreSS classifies more proteins correctly. In this case, 94 % of the proteins are correctly classified into their superfamilies. Our method performs the worst when ϵ_t = 1.0. This result leads to two important conclusions: 1) Searching by sequence information alone is worse than searching based by sequence and structure simultaneously. 2) For purposes of classification, our extraction of feature vectors for sequence is better than those for structure.

Figure 6.8 plots the effect of the window size on the classification quality of ProGreSS. The best results are achieved at w = 3. At this window size, ProGreSS can classify 100 % and 97 % of the classes and superfamilies correctly. ProGreSS

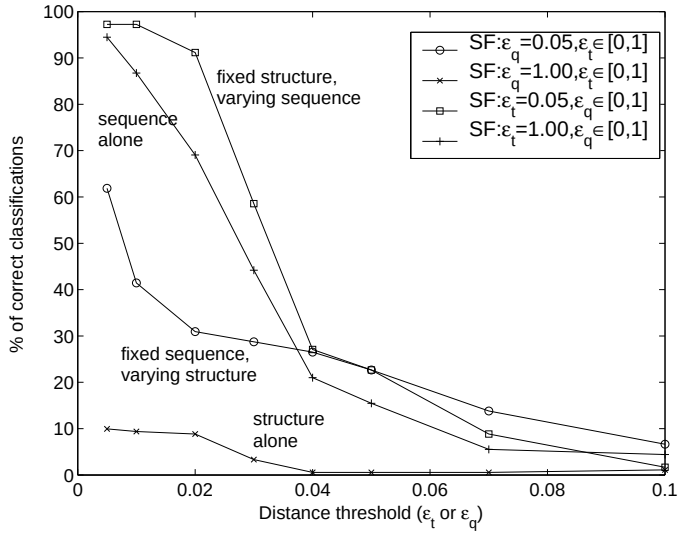

Figure 6.7: Percentage of query proteins correctly classified for different values of ϵ_t (ϵ_q) when ϵ_q (ϵ_t) is fixed.

performs worse for smaller window sizes since correlations between the consecutive residues are not reflected to the index structure. As w becomes larger than 3, ProGreSS starts to miss some of the good results since shorter local matches are not preserved for large w.

Finally, Figure 6.9 compares the accuracy of our technique with CTSS. We show the number of correct proteins (those from the same superfamily as the query protein) for different values of c. CTSS finds 3 out of 10 correct proteins in the first 100 candidates. On the other hand, our method finds the same number of proteins within the first 4 candidates.

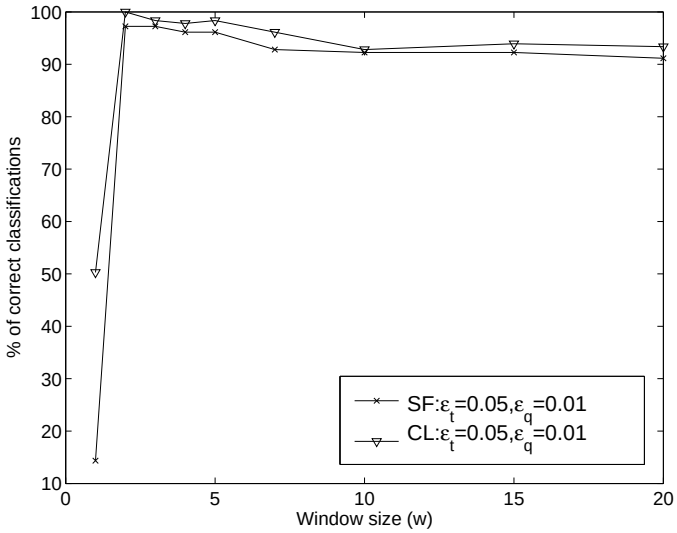

Figure 6.8: Percentage of query proteins correctly classified for different values of w.

6.4.2 Performance test

In this experiment set we compare the performance of our method to CTSS. In order to have fair results, as described in Chapter 5, we run CTSS in two phases: 1) the top c candidates are found using the original CTSS code and each candidate is aligned to the query by using SW based on its structure score matrix. 2) The optimal sequence alignment of all the database proteins to the query are determined using SW alignment. For CTSS and ProGreSS, we choose c = 100 and 4 respectively. This is because the quality of their candidates are similar for these values of c (see Figure 6.9). We run queries for all of the 181 proteins and align only the candidate proteins to each of the query proteins.

Figure 6.10 shows the average time spent by CTSS and our method. The run

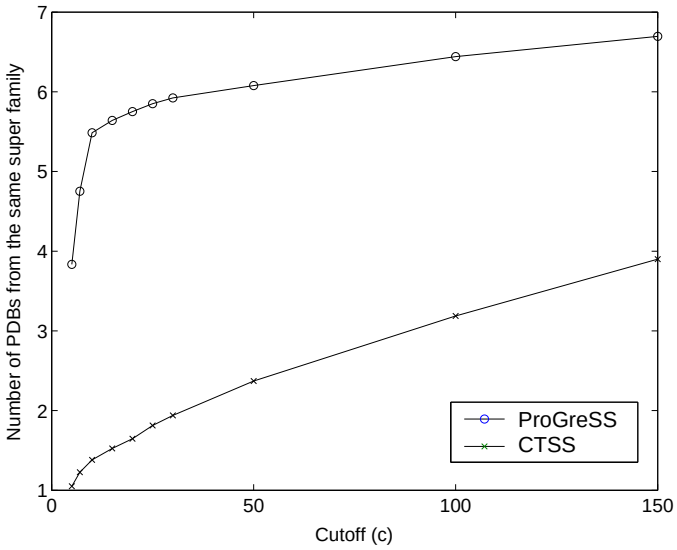

Figure 6.9: Number of proteins found from the same superfamily as the query protein for ProGreSS and CTSS for different values of c.

times for CTSS and SW are 38 and 18 seconds respectively. The graph for CTSS+SW is flat since these methods are independent of η. ProGreSS runs faster than CTSS+SW for all values of η. For η = 10, ProGreSS runs in only 1.5 seconds (i.e., 37 times faster than CTSS+SW). As η gets smaller, ProGreSS runs slower. This is because when a query box partially overlaps a cell, more MBRs are tested for intersection. As η becomes larger than 10, the performance of ProGreSS drops since the total number of MBRs in the index structure increases.

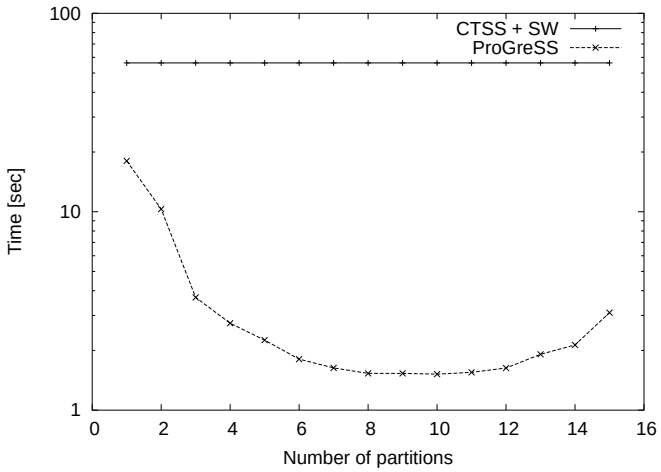

Figure 6.10: Comparison of running times of ProGreSS and CTSS+SW.

6.5 Discussion

In this chapter, we considered the problem of joint similarity searches on protein sequences and structures. We proposed a sliding-window–based method to extract feature vectors on the sequence and structure components of the proteins. Each feature vector is mapped to a point in a multi-dimensional space. We developed a novel index structure by partitioning the space with the help of a grid, and clustering these points using Minimum Bounding Rectangles (MBRs). Our search method finds the number of feature vectors that are similar to the feature vectors of a given query for each database protein. We also proposed a new statistical method to compute the significance of the results found at the index search phase. The results are sorted according to their significance and the most promising results are aligned using the Smith-Waterman (SW) method [84] and the least-squares method by Arun *et al.* [5]

to find the optimal alignment.

According to the experimental results on a set of representative query proteins, ProGreSS classified all of the classes and 97 % of the superfamilies correctly. ProGreSS ran 37 times faster than CTSS, an efficient structure search technique described in Chapter 5, combined with the SW technique for sequences.

Combined learning from multiple data sources is an important research problem since each data provides a correlated yet different type of information about the protein. ProGreSS provides the user a wide flexibility of search parameters to assign weights on each of these data types. We believe that, the methods discussed in this chapter are an important step in understanding the functions of proteins better, and will be widely applicable in the area of proteomics.

Chapter 7

Automated Protein Classification Using Consensus Decision

A global view of the protein structure universe can be established with the help of structural classification databases [44, 68, 64]. Classification databases are used to define the relationships—in terms of sequence, structure, and function—of proteins. Of these classification schemes, SCOP [64] is created mainly by manual inspection. This is perhaps the reason that it is accepted by many researchers as the most accurate classification scheme (or the ground truth). However, SCOP is updated every six months, since it is quite a labor intensive process to manually place a protein structure into the correct category in a hierarchical classification of about 25K protein structures as of July 2004. Furthermore, the current 100 protein per week growth rate means about 2600 protein structures in six months. Therefore, if one requires a dynamic, up-to-date view of the protein structure universe in a timely manner, accurate automated classification techniques should be developed to aid in manual classification process.

In this chapter, we propose a novel technique for automatically generating the SCOP classification of a protein structure with high accuracy. High accuracy is achieved by combining the decisions of multiple methods using the consensus of a committee (or an ensemble) classifier. Our technique is rooted in machine learning which shows that by judicially employing component classifiers, an ensemble classifier can be constructed to outperform its components. We use two sequence- and three structure-comparison tools as component classifiers. Given a protein structure, using the joint hypothesis, we first determine if the protein belongs to an existing category (family, superfamily, fold) in the SCOP hierarchy. For the proteins that are predicted as members of the existing categories, we compute their family-, superfamily-, and fold-level classifications using the consensus classifier. We show that we can significantly improve the classification accuracy compared to the individual component classifiers. In particular, we achieve error rates that are 3-12 times less than the individual classifiers' error rates at the family level, 1.5-4.5 times less at the superfamily level, and 1.1-2.4 times less at the fold level.

7.1 Introduction

A global picture of the protein universe is necessary to gain a better understanding of how proteins function. Numerous classification schemes have been developed for defining the relationships—in terms of sequence, structure, and function—of proteins. Protein classification schemes employ different heuristics, similarity metrics, and different degrees of automation [44, 68, 64]. Of these classification schemes,

SCOP [64] is created mainly by manual inspection. This is perhaps the reason that it is accepted by many researchers as the most accurate classification scheme (or the ground truth).

With the exponential growth in the number of newly-discovered protein structures, the view of the protein universe is constantly changing. In order to understand the functions of proteins and their relationship to each other, classifications of proteins should be updated frequently. SCOP, being built by labor-intensive visual inspection, is updated only every 6 months. On the other hand, automated classification schemes have the advantage that the view of the protein universe can be updated frequently to include newly-discovered protein structures in a timely manner. Hence, there have been efforts to infer the SCOP classification of a protein structure by using the results of an automated method [32]. However, the validity of such an approach is bound by the accuracy of the method employed. *Our research is hence geared toward providing timely and accurate SCOP classification results in an automated manner.*

Our approach is rooted in consensus building in machine learning, which shows that an ensemble classifier with a better performance can be constructed as a committee of component classifiers [23, 61, 78]. A similar idea has been applied to the fold recognition problem by Lundström et al. [59]. By using a neural-network-based consensus predictor they were able to improve the prediction accuracy. However, their goal in fold recognition was to predict the *structure* of a protein sequence rather than its SCOP classification. Furthermore, using only the sequence information is insufficient for ensuring accurate SCOP classifications, particularly for remote homologs.

The problem of automated generation of SCOP classification has been shown to be a difficult one. In a recent study, Portugaly and Linial estimated the probability of a protein sequence to have a new fold [71]. Of the protein sequences that they assigned a 90% probability to have a new fold, only half of them actually had new folds. In another work, Lindahl and Elofsson compared several sequence-comparison methods for classification of protein sequences at the family, superfamily, and fold levels [56]. They observed that the best-attained accuracy dropped from 75% to 29% from the family to the superfamily level and down further to 15% for the fold level. After analyzing the results of several methods, the authors concluded that a combination of methods may improve the performance. However, they reported such an improvement was not possible as they achieved only limited success.

Nonetheless, our goal is different: prediction of SCOP classification of a newly-discovered protein structure. Both sequence and structure information are available for this task, and we show in this chapter that an ensemble classifier can outperform individual components if (1) a correct information-aggregation framework is used and (2) the right component classifiers are employed. In particular, we show that it can be beneficial in using a consensus decision framework, which is grounded in machine learning, with component classifiers that address both sequence and structure information for predicting accurate SCOP classifications for proteins with known structures. We employ a *decision tree* approach to combine classification decisions made by two sequence-based, and three structure-based classifiers. Given a recently-solved protein structure, we are able to to predict its SCOP classification with high accuracy, using a consensus decision made by these five classifiers. *What is signif-*

icant is that the boosted accuracy numbers are close to the theoretically maximum performance achievable using such a consensus building framework.

The remainder of this chapter is organized as follows. In Section 6.2, we define the problem and describe how a sequence/structure similarity search method can be used as a classifier. In Section 6.3, we review the sequence- and structure-based classifiers we have used in our framework, and compare their relative consistency In Section 6.3.2 we analyze the classification performance of individual methods. We show that it is possible to develop a better classifier with higher accuracy by combining the decisions of individual methods. In Section 6.4, we propose a method for building a consensus classifier based on the idea of decision trees from machine learning. In Section 6.5, we evaluate the performance of our algorithm using different versions of the SCOP. We conclude with a brief discussion.

7.2 Problem Definition

The main questions to be answered when classifying a novel protein structure are:

i. Does this protein belong to an existing category (family/superfamily/fold) in SCOP hierarchy, or does it need a new category to be defined?

ii. If this protein belongs to an existing category, what is its classification (label)?

The SCOP database uses a four-level taxonomy: class, fold, superfamily, and family. Each *domain* in a protein structure is assigned to one category in each of

these four levels. In this chapter, for the sake of uniformity, we have used single domain proteins. Therefore, the word *protein* is used instead of *domain*. The top level of SCOP, *class*, is defined by the number of secondary structures in the proteins and their general layout. Since the class label can be assigned automatically and the assignment does not present a significant challenge, we do not consider it in our classifications.

At the family level, proteins are assigned to the same SCOP family if they have a high sequence identity ($\geq$30%), or they perform similar functions but have a relatively lower sequence similarity ($\geq$15%). So the main similarity measure at the family level is sequence similarity. Thus, we would expect that sequence-comparison tools are more appropriate than structure-comparison tools as component classifiers for automated SCOP classification at the family level. At the superfamily level, though, distant similarities are considered. Proteins in the same superfamily probably have evolutionary relationships that are inferred by structural similarities rather than sequence similarities. Hence, we would expect structure-comparison tools to be more successful than sequence-comparison tools at this level. The fold level is the most blurry level of all. At this level, proteins are grouped together not because they show significant similarity, but because they share similarity in the arrangement of their secondary structures. Remote structure similarity is the major similarity metric at the fold level. Thus, structure-comparison tools are expected to outperform sequence-comparison tools at this level.

These three levels of classification are not independent of each other. We have a hierarchical scheme where each protein is classified into a family which in turn

belongs to a superfamily that is a subclassification of the fold category. Assigning a superfamily to a protein which possesses more than a 30% sequence identity to some of the proteins in the database is a trivial task, since we can accurately assign a family to that protein (based on sequence similarity). Superfamily and fold assignments are inherent in this assigned family. In order to extract true performances at different levels, we have to make a complete separation of different levels in the classification, as in [56]. For example, at the superfamily level only the proteins without family-level relationships are queried, and the family/superfamily-level relationships are ignored when evaluating fold-level performance.

For a given query protein structure we proceed to find its SCOP classification in a bottom-up manner. The ensemble classifier first tries to assign a family label to the query protein. If it is successful, the classification process is complete. Otherwise, the query protein needs a new family category to be defined in SCOP, and its superfamily category is sought instead. If a superfamily cannot be assigned, the ensemble classifier proceeds to the fold level. The protein is predicted to have a new fold, if the ensemble classifier is not able to assign a category in any of the three levels.

7.2.1 Building a component classifier using a comparison tool

Each component classifier is trained to answer the first question, i.e., whether a query protein belongs to an existing category. We train the classifier using the procedure outlined below: For each query protein p in the query set QS, we find the closest protein, i.e., the nearest neighbor, in the database of proteins with known classification, based on the similarity criteria defined by the comparison tool, T, that

the classifier uses. The similarity score assigned by the comparison tool provides a measure of distance between the query protein and the whole database, e.g., a very low score indicates that the query protein does not possess significant similarity to any of the database proteins.

We sort the query proteins using these similarity scores. The goal of the classifier is to partition the query set into two in such a way that one partition contains all the query proteins that belong to existing categories, and the other partition contains all the proteins that need a new family/superfamily/fold label. The partitioning is achieved through a score cutoff. However, a perfect partitioning is usually not possible due to the blurry boundaries of categories. For each tool, we determine the best score cutoff as the one that *maximizes* the number of correct predictions. If the score of the alignment for the query protein is greater than the cutoff, then the classifier is construed to predict that p belongs to an existing category; otherwise, p should belong to a new category.

For the second problem of label assignment, we use the 1NN (first nearest neighbor) classification method. In this method, T finds the nearest neighbor to the query protein (most similar protein) in a database, and the classifier assigns the query protein the same label (family, superfamily, or fold) as that of its nearest neighbor.

7.3 Classifiers Used in Our Ensemble Scheme

We use two sequence classifiers. The first one is a model-based sequence comparison tool, HMMER [24], for comparing a protein sequence against models of the

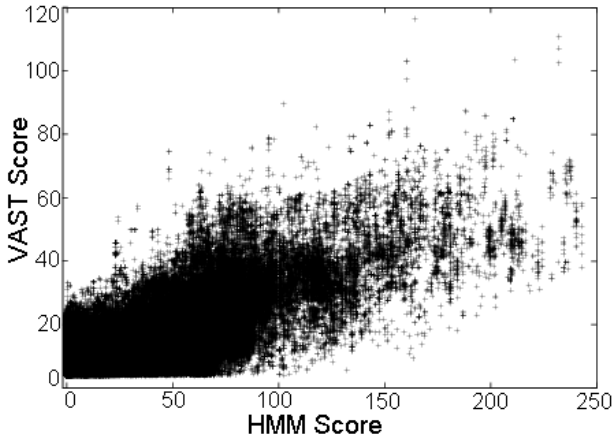

Figure 7.1: Comparison of HMM and Vast scores.

SUPERFAMILY database [36]. This database is a hidden Markov model (HMM) library representing all proteins of known structures. It is manually curated to classify proteins at the SCOP *superfamily* level. HMMER assigns a similarity score to the sequence according to its match with a model. In the rest of the chapter, we will refer to this method as HMM.

The second sequence classifier we use is PSI-Blast [2]. PSI-Blast is an improved version of Blast that works in iterations. In the first iteration, Blast is run and a new scoring scheme is created based on the set of close neighbors. In the ensuing iterations this new scoring scheme is used and updated as new close neighbors are found.

We use three structure classifiers. The first one, CE [80], is a pairwise structural alignment tool. It uses inter-atomic distances of C_α atoms to find small (8 residues), geometrically-similar fragments from the pair of proteins. Then, CE combines these

fragments to form longer matches.

We also choose Dali [42], a structure-similarity comparison tool, as one of our classifiers. Dali computes the distance matrices of two proteins and then finds the alignment by a Monte Carlo algorithm. Dali has been used to create FSSP [44], an automated protein classification database.

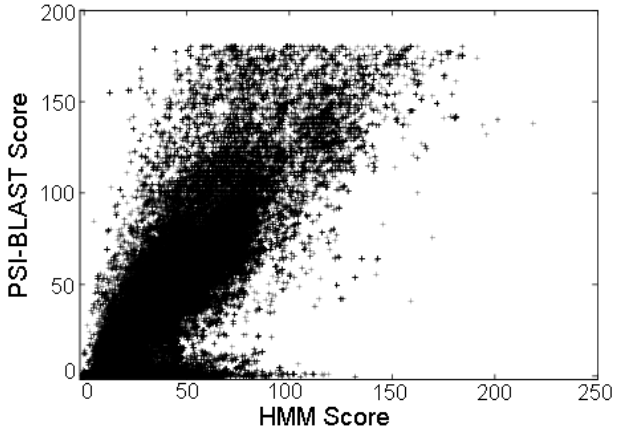

Figure 7.2: Comparison of HMM and PSI-Blast scores.

The final classifier we have chosen is Vast [60]. It is designed for identifying remote homologies. It uses secondary structure elements to locate initial matches. These properties distinguish it from other structure-comparison tools, since Vast prefers small biologically meaningful matches over optimal global alignment.

7.3.1 Relationship between the classifiers

Each sequence- and structure-comparison tool described in the previous section assigns a score for a pair of proteins, that indicates the statistical significance of the

similarity between them. In particular, we have used the z-scores reported by CE and DALI, p-values reported by VAST, and e-values ($-log$(e-value)) reported by HMM and PSI-Blast as similarity scores. Below, we explore the correlation between the scores of different methods. Figure 7.1 shows the correlation between the scores of HMM and Vast. Each data point in the 2D plot represents a pair of proteins, and the point coordinates are the comparison scores from HMM and Vast, respectively. A perfect correlation would consist of entries on a straight line. In this case, we see quite a bit of disagreement. This is expected since we are comparing the scores of a sequence-comparison method with those of a structure-comparison method. However, Figures 7.2 and 7.3 depict that there is considerable disagreement between two structure- or two sequence-comparison methods as well. In a similar study, Shindyalov and Bourne [81] compared CE and Dali scores and showed that there were many proteins that were found similar by CE and dissimilar by Dali, and vice versa. Our solution to this dilemma is to reconcile the discrepancies between the classification tools by combining them into a consensus decision framework.

7.3.2 Performance of component classifiers

We performed a number of experiments to understand the individual performance (accuracy) of the tools when they are used as component classifiers. In these experiments we assumed classifications of all the proteins in SCOP v1.59 ($DS159$) are known. We then classify the new proteins introduced in SCOP v1.61 ($QS161$) into families, superfamilies, and folds by using structure- and sequence-comparison tools. As described in Section 6.2, to get the true performances of classifiers at dif-

ferent levels, we need to separate the query set based on levels of similarity. At the family level all new proteins introduced, ($QS161$), are queried. At the superfamily level, only proteins, ($QS161_{\text{newFam}}$), that do not have family-level similarities are queried. And at the fold level, only proteins ($QS161_{\text{newSF}}$) that do not have family/superfamily-level similarities are queried.

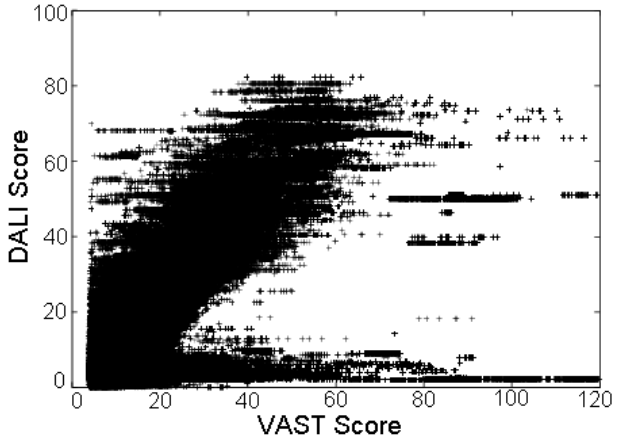

Figure 7.3: Comparison of Vast and Dali scores.

Recognition of members of existing categories

The performance results for each tool are shown in Figure 7.4. At the family level, we can clearly see that the sequence tools outperform the structure tools by achieving 94.5% and 92.6% accuracy. The success rate of the structure tools is only 89%. Figure 7.4 also indicates that it is possible to manage higher success rates by combining the tools. The sixth column of the figure shows the aggregate accuracy of the tools. For 83% of the query proteins, all five tools make correct decisions (as

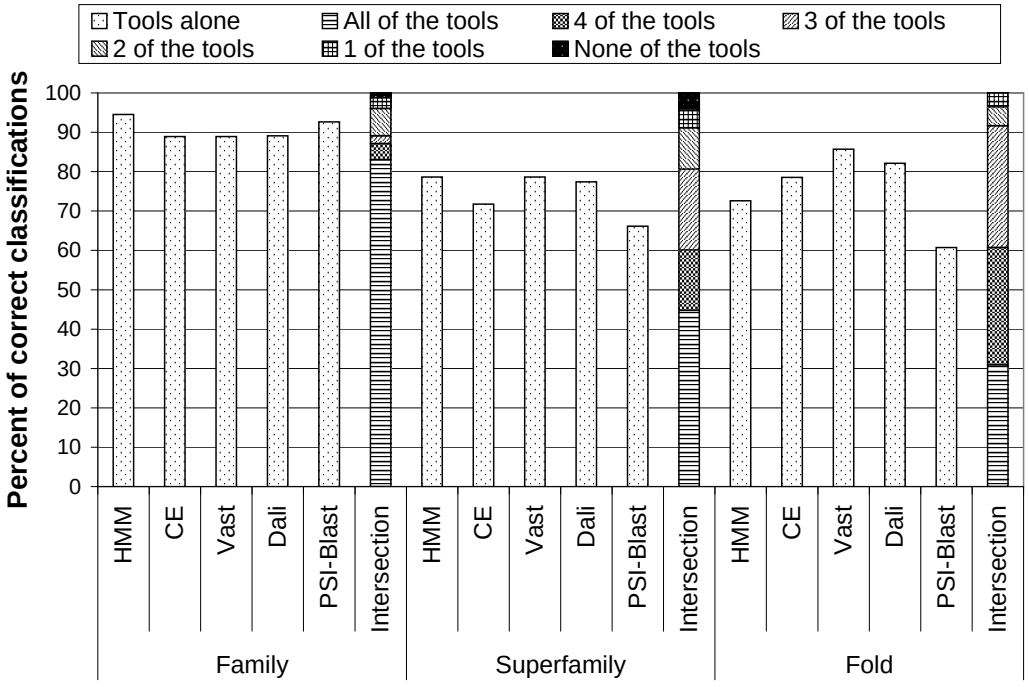

Figure 7.4: Performance of individual classifiers on the membership problem for the new proteins introduced in SCOP v1.61.

to whether the query protein belongs to a category), for 4.1% of the queries 4 tools, for 1.9% of the queries 3 tools, for 6.9% of the queries 2 tools, and for 2.7% of the queries only one tool makes the correct decision. An interesting point here is that for 98.2% of the proteins, at least one tool is successful. So, it is theoretically possible to classify up to 98.2% of these proteins correctly by combining the results of the individual tools.

At the superfamily level, the structure tools match or better the performance of sequence tools, as expected. However, the overall performance of the tools drops significantly from the family level. This is expected since classification at the family level is the easiest. HMM has one of the best performances, and this is no surprise

considering that HMM was manually tuned for superfamily classifications. Among the structure tools, Vast has the best performance with a 78.6% success rate. PSI-Blast performs poorly with a success rate of only 66.1%. Only 44.7% of the queries can be classified correctly by all five tools. For 4% of the queries, none of them is successful.

Structure tools clearly outperform sequence tools at the fold level. Vast has the best performance with a 85% success rate. PSI-Blast again has the worst performance with a 60.7% success rate. For only 30.9% of the queries, all five tools are successful. An interesting aspect is that all the query proteins are classified correctly by at least one tool. This again raises the possibility of achieving better accuracy through a combination of the results.

Classification assignment

Once a tool has marked a new protein as a member of an existing category, the classification of query protein is complete, i.e., the query protein is assigned to the same category as its nearest neighbor. The next question is to judge the accuracy of this assignment, i.e., whether the assigned category is the correct one. The accuracy results for the different tools are shown in Figure 7.5. The results are reported for the list of query proteins that are known to be members of existing families, superfamilies, and folds at each level respectively.

The accuracies of the tools are high at the family level. All except Dali have success rates above 90%. HMM has the best performance with 94.8% accuracy and is followed by Blast with 92.3% accuracy. For 76.5% of the queries, all five tools

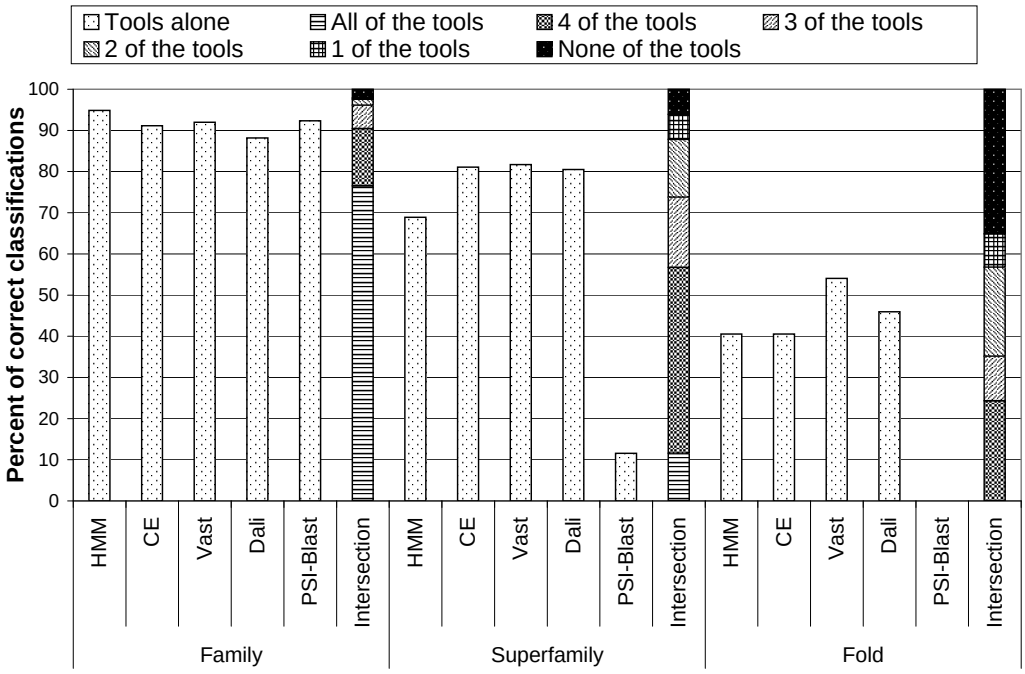

Figure 7.5: Performance of individual classifiers on category assignment problem for the new proteins introduced in SCOP v1.61.

are able to assign the correct family label. For only 2.1% of the queries, none of them is successful.

At the superfamily level, the structure tools perform better and achieve 80.4-81.7% success rates. The poor performance of sequence tools is surprising since sequence similarities still matter at this level. HMM, especially, is expected to perform well, since it is manually tuned for this level. For 6.1% of the queries, none of tools is able to assign the correct superfamily label. This means one can potentially achieve a theoretically maximum 93.9% success rate by combining the results of these five tools. This is quite high compared to the performance of the best tool, 81.7%.

At the fold level, all tools seem to perform poorly. Note that the number of test proteins decreases significantly at this level (37 proteins), because the proteins with family/superfamily relationships are discarded. Therefore, wrong classification of even a single protein has a strong effect (3%) on the accuracy results. At the fold level, PSI-Blast is not able to make even one correct fold assignment whereas Vast assigns correct folds to 54% of the queries. For 35.1% of the queries, none of the tools is able to assign the correct fold label. This high failure rate suggests noise from similar folds, and is investigated further.

7.4 Automated Classification Using Ensemble Classifier

As evident from our initial experiments, an ensemble classifier can potentially obtain higher classification accuracy than any single component classifier. There are many studies in the area of machine learning and pattern recognition that address the intelligent design of ensemble classifiers [23, 61, 78]. These include both competitive models (e.g., bagging and arcing) and collaborative models (e.g., boosting). Our method employs a hierarchical decision tree to answer the question whether the query protein belongs to an existing category. If the answer is yes, we then use Bayesian decision rules to assign the protein an appropriate label.

Since our goal is to classify protein structures at three levels of hierarchy in SCOP, a hierarchical classification algorithm is appropriate. An overview of the general algorithm is given in Figure 7.6. Here, parameter p is the protein whose

classification is sought and R is the set of classification rules found during training.

7.4.1 Normalization of similarity scores

In order to combine the results from different tools, we need to find a way to normalize their results into a consistent scale. We achieve this through *binning*. A bin is a *tool-neutral* accuracy extent, e.g., 90%-100%, 80%-100%, instead of a *tool-specific* similarity score. The bins are manually crafted to obtain the best spatial resolution and every classification tool produces its own set of bins. The procedure used is as follows: We use $DS159$ as the database and $QS161$ as the query set (See Table 7.2). For each protein in $QS161$, we record the top similarity score in $DS159$ (nearest neighbor) using a particular comparison tool (or a component classifier). Performing this operation for all the proteins in $QS161$ results in a histogram indexed by the similarity score used by the tool. We then sort the scores and scan, from high score (or similarity) to low score, to decide on the bin boundaries. The bins partition the score space of each tool into buckets, each containing roughly the same number of proteins. The kth such bin corresponding to tool i is called E_{ik} (E stands for *Existing*).

We also construct a dual set of bins for each tool from the lower end of the score space, based on the tool's accuracy of predicting that the protein is new (i.e., does not belong to an existing category). The kth such bin corresponding to tool i is called N_{ik} (N stands for *New*). Note that the above bin construction is repeated at each of the three levels: family, superfamily, and fold.

Given a query protein, the score computed by a classification tool can be used

to map it to a pair of E and N bins, say (E_i, N_j). The classification tool assigns a single confidence level, c, to this pair of bins using:

$$c = \frac{|p : p \in E_i \wedge p \in N_j \wedge p \text{ is existing}|}{|p : p \in E_i \wedge p \in N_j|} \times 100 \tag{7.1}$$

where p represents the training proteins. This process is repeated for each tool, resulting in a vector of confidence levels for the query protein. Next, we discuss how the vector of values is combined together through decision tree rules and weighted Bayesian rules.

Input: p is the query protein for classification, R is a set of classification rules.
Algorithm FIND-CLASSIF(p, R)
If HAS-CLASSIF$(p, R, family)$
    return ASSIGN-CLASSIF$(p, R, family)$
elseif HAS-CLASSIF$(p, R, superfamily)$
    return ASSIGN-CLASSIF$(p, R, superfamily)$
elseif HAS-CLASSIF$(p, R, fold)$
    return ASSIGN-CLASSIF$(p, R, fold)$
else return *New-Fold*

Figure 7.6: Overview of the classification algorithm.

7.4.2 Recognition of new categories using decision trees

The rule set R used in Figure 7.6 affects the performance of the classification algorithm. Intuitively speaking, if rules in R can assume very general and complicated forms, then one can expect an increased likelihood of finding a rule that generates good classification results. However, complicated rules with many tunable param-

eters result in large pools of candidate rules, which increase training time and size of the training samples needed. This general trade-off, often referred to as the bias-variance trade-off [23], is well understood in the machine learning community. It states that flexible, general rules provide better (small bias), but less predictable (large variance), classification results. The danger of over-fitting is also much higher with flexible rules.

In our framework, we have five component classifiers whose results can be combined to form rules in R. The decision of each component classifier can be considered as an attribute of the query protein. Using the decision tree approach [23], the final decision about a protein is made by consulting a set of attributes in a hierarchical manner.

However, even with only five component classifiers, the number of different decision trees one can construct is huge. There are several parameters that affect the decision tree creation process. The first one is the branching factor at each node of the tree. That is, the decision consulted at each node may *split* the training data into several subsets.

Fortunately, for the problem of SCOP classification by combining sequence/structure comparison tools, a biologically sound decision can be made about the branching factor at each node. That is, at each level by consulting the confidence levels, we can split the training data into *three*: one partition being query proteins that belong to existing categories (because of strong evidence of similarity), another partition being query proteins that are new (because of strong evidence of dissimilarity), and the other partition being query proteins that are in the twilight zone. This observation

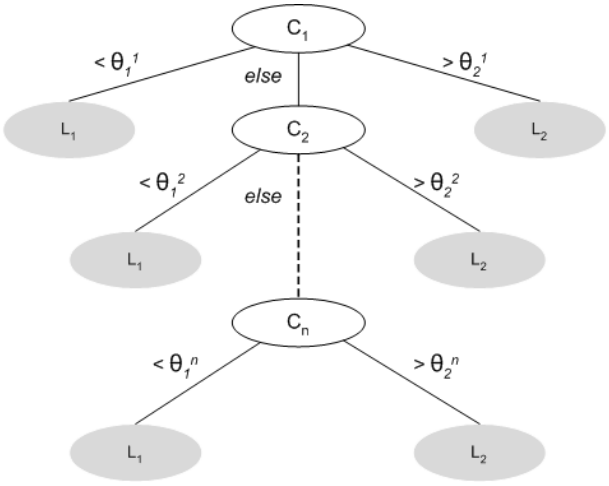

Figure 7.7: The general structure of the decision trees suitable for protein classification purposes.

limits the search space to decision trees having the structure shown in Figure 7.7, where C_i is the combined confidence level of a number of tools, θ_j^i is the confidence threshold used to assign label L_j to a query protein at level i. L_1 and L_2 are the labels for proteins that need new family/superfamily/fold categories, and that belong to existing family/superfamily/fold categories respectively. Note that, there is a single threshold at the last level, i.e., $\theta_1^n = \theta_2^n$.

We further restrict the tree structure to be at most three levels because of the small number of component classifiers used and the desire to expedite the training process. Even with this fixed decision-tree structure, there are two important sets of parameters that should be determined, C_is and θ_j^is. These parameters are the most crucial components of the decision tree since the classification accuracies vary remarkably for different choices.

Terms C_is can assume many different forms, and some popular ones are the sum,

product, max, and min rules, which compute the sum, product, max, and min of the component scores, respectively. The search space is exceedingly huge without some restriction on the form that C_is can assume. In our experiment, we consider only the sum rules, i.e., the decision is based on the weighted sum of the confidence levels of component classifiers. To further simplify the analysis, each C_i uses the confidence levels of up to three components and always weighs the component scores equally.

Automated decision tree construction

We generated all possible trees with C_is composed of confidence levels of at most three tools. There are $15,625$ such trees. The accuracy of each tree is determined by examining the categories of the proteins at the leaf nodes. The best accuracy is achieved if all the proteins that are labeled as L_1 are actually new proteins, and all the proteins labeled as L_2 actually belong to existing categories. The erroneously labeled proteins at leaf nodes decrease the accuracy. In other words, to achieve high accuracies we want the leaf nodes to be as pure as possible, containing only proteins sharing existing labels or needing new labels, both not both. In the machine learning community, this is referred to as minimizing the impurity. Various impurity functions can be used [23]. Most commonly used impurity measure, *entropy impurity* is defined as:

$$i(N) = -\sum_j P(w_j) \log_2 P(w_j), \tag{7.2}$$

where $P(w_j)$ is the fraction of patterns at node N that are in category w_j.[1]

[1] In our case, there are two categories: one corresponds to the proteins that can be assigned an existing label and the other corresponds to the proteins that need a new label.

To search for the best values for θ_j^i for each tree, we can consider a global optimization by minimizing the overall impurity (summation of the impurities at each leaf node). However, this is a five dimensional optimization problem (to determine the five thresholds in Figure 7.7) for each of the $15,625$ trees, which is not feasible. Instead, we can use a greedy approach that does not guarantee optimality but provides efficiency.

In our approach, the best thresholds at a level are determined by examining only the leaf nodes at that level. However, we need to design this strategy carefully, because trying to minimize impurities using a local, greedy procedure usually produces trees with bad accuracy. The reason is that one simple way of achieving low impurity at a certain level is to limit the decisions for a small portion of the training data. E.g., if we pass all but two training samples (one sharing an existing label and the other needing a new label) down the middle tree branch in Figure 7.7, and assign the two samples to the appropriate left and right branches, we obtain zero impurity at this level. However, this greedy strategy causes most of the decisions to be made at the bottom of the tree where most of erroneous labeling occurs. To overcome this problem, the cost function must be augmented to balance the impurity and the number of samples classified at a particular level.

Using this greedy approach, we could generate very accurate ensemble classifiers that perform much better on the training data than the component classifiers used. However, when we tested these decision trees on the test data set, their performance dropped drastically. The main reason, we suspect, is that the greedy approach tends to over-fit the training data. We believe that this again can be attributed to the

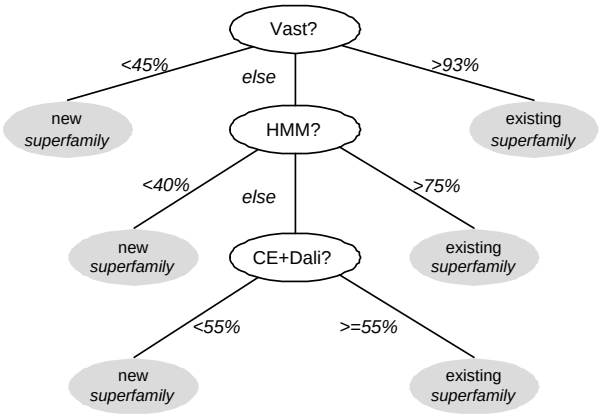

Figure 7.8: The decision tree for recognition of proteins that belong to existing superfamilies.

bias-variance tradeoff. I.e., a decision tree with many tunable parameters can perform well on training data, but not on validation data.

Manual decision tree construction

As a solution to the over-fitting problem, we tried to construct decision trees manually based on lessons learned from Section 7.3 on the strength and applicability of the component classifiers. First, the C_is are determined by referring to Figure 7.4. The tools with the highest performance are chosen to perform at the first level. Consequently, for each following level the tool or tool combination that can best classify the remaining proteins is chosen.

After the tool combinations that perform at each level are decided, the confidence thresholds for decisions, θ_j^is, are chosen. The most important benefit of manual decision tree construction surfaces in choosing these thresholds. We make use of

the knowledge of the performance of the tools that are used at each level, as we did for determining the C_is. This knowledge allows us to optimize the impurities of leaf nodes at two levels simultaneously. We find the four thresholds such that:

$$\min_{\substack{\theta_1^i \quad \theta_2^i \\ \theta_1^{i+1} \quad \theta_2^{i+1}}} \sum_{j=1,2} \delta(L_j^i) + \delta(L_j^{i+1}) \tag{7.3}$$

Here, θ_1^i and θ_2^i are confidence level thresholds for leafs L_1^i and L_2^i at level i. $\delta(L_j^i)$ is a function proportional to the impurity of the leaf L_j^i and inversely proportional to its population.

Therefore, local optima of two levels are computed simultaneously. This heuristic is closer to the global optima than the one-level greedy approach. Equation 7.3 is a 4 dimensional optimization problem where each dimension θ_j^i can be optimized to get the best impurities at the leaf nodes. However this leads to the over-fitting problem explained previously. So, we determine manual thresholds that are between the score clusters as shown in Figure 7.9. A_1 and A_2 are the thresholds found by the automated greedy approach. M_1 and M_2 are the manual thresholds. It is evident from the figure that A_1 and A_2 over-fit the data, whereas M_1 and M_2 do not. Moreover, the benefits of such procedure is twofold. First, the over-fitting of the training data is avoided. Second, since the number such significant thresholds is limited, the search space of Equation 7.3 is bounded by a discrete minimization problem.

The manually constructed decision trees vary with the taxonomy level. When deciding if a protein is from an existing family, we give priority to the sequence tools. So, we first assign a family label based on HMM and PSI-Blast. For our training set,

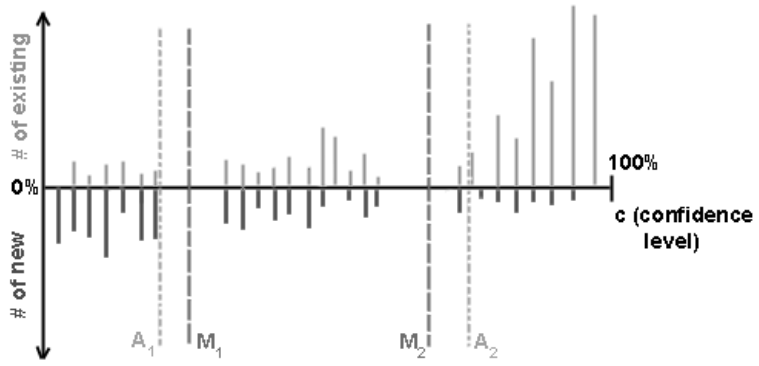

Figure 7.9: An example histogram of the confidence levels for the training data.

these two tools together are able to assign 90% of the queries confidently. For the remaining proteins, we use the structure tools. An interesting point here is that if sequence tools are unable to find a significant match for a protein, but structure tools find a significant match, then the protein likely belongs to a new family.

At the superfamily level, Vast and HMM perform the best, as seen from Figure 7.4. So, they have the top priorities. We first use Vast, and then use HMM on the proteins that Vast cannot predict with confidence. Finally, on the twilight zone of these two tools, we apply Dali and CE. At the fold level, structure tools perform better. So, we use the structure tools in the order of Vast, CE, and Dali. The complete set of decision tree rules, for family, superfamily, and fold levels from top to bottom rows, is shown in Table 7.1. At each level (column), a combination of tools is run and the probability of being a member of an existing category is assigned to each protein. The proteins that have probabilities higher than the indicated range are assigned to the predicted category, the ones within the range are passed to the next

Level 1	Range	Level 2	Range	Level 3	Threshold
HMM+Blast	(60%:95%)	CE+Vast+Dali	(70%:85%)	HMM+Blast	85%
Vast	(45%:93%)	HMM	(40%:75%)	CE+Dali	55%
Vast	(50%:85%)	CE	(80%:90%)	Dali	60%

Table 7.1: Heuristic decision tree rules for the category membership problem.

step, and those below the range are deemed new. For the last level, only a single threshold exists.

7.4.3 Classification assignment for members of existing categories

After deciding that a protein is a member of one of the existing categories, we assign its classification. The assignment is done by using a weighted Bayesian rule, in which the weights for the components classifiers are determined according to their training accuracies. Each tool assigns the query protein to a category with a certain confidence. These categories are compared and the query protein is assigned to the category that gets the highest probability. We consider the reliability of each tool to weigh its contribution to the consensus decision. Figure 7.5 provides us this information. When assigning family labels, sequence tools are more reliable than structure tools, and when assigning superfamilies and folds, structure tools are more important than sequence tools.

7.5 Experimental Evaluation

To validate that consensus decision indeed improves classification performance, we apply the standard validation technique in pattern recognition [23]. Two data sets are used: a training set and a test set. In each case, we need a set of database proteins (proteins of known classification) and query proteins (proteins to be assigned an existing fold/superfamily/family label or a new label). We choose three versions of the SCOP database [64] in our experiments, thus also reflecting the expansion of the protein structure universe. SCOP version 1.59 ($DS159$, May 2002) and version 1.61 (December 2002) are used to generate the training set. The training query set is generated by extracting the protein chains from version 1.61 that were introduced after version 1.59, $QS161$. After training our method using $DS159$ and $QS161$, we apply the same strategy to SCOP versions 1.61 and 1.63 (May 2003) to evaluate the performance of our algorithm. Table 7.2 shows the number of protein chains used for training and validation. As seen in the table, the distributions of introduced superfamilies and folds changed dramatically between versions 1.61 and 1.63. This change poses a challenge to our ensemble classifier, since the generated classification rules depend on the training set. However, as the results of our experiments show below, our algorithm performs well in this real-life setting and the difference between the classification accuracy of training and classification accuracy of evaluation tests is in agreement with the difference between training error and generalization error observed in other pattern recognition contexts [23].

7.5.1 Training Procedure

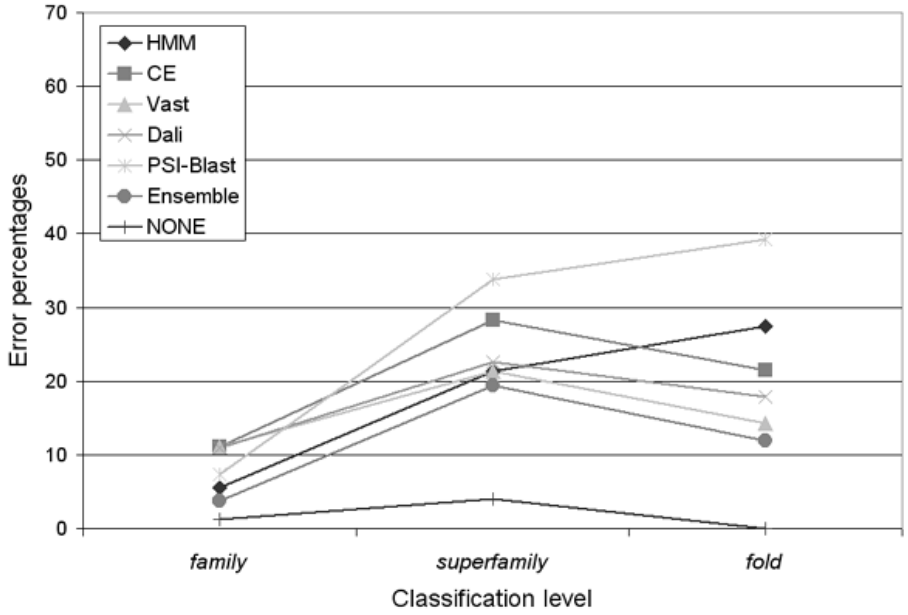

Figure 7.10: Performance of individual classifiers compared to the ensemble on category membership problem for the new proteins introduced in SCOP v1.61.

The ensemble classifier is trained in a similar hierarchical manner described in Section 6.3. The first case is to recognize if a new protein has an existing classification. We train the ensemble classifier, i.e., produce the decision tree rules, to perform best on the database $DS159$, proteins in SCOP v1.59, with the queries introduced in v1.61, $QS161$. Figure 7.10 depicts the comparison of the ensemble with the five component classifiers. At the family level, the best performance among the tools is achieved by HMM with a 5.5% error rate. The ensemble is able to reduce this error rate to 3.7%. Figure 7.4 shows that for 1.2 % of the queries none of the tools find the correct classification. So, one cannot reduce this error rate below 1.2%, which can

be accepted as the theoretical limit. The ensemble manages to outperform HMM 1.5 times, PSI-Blast 2 times, and the structure tools more than 3 times. The ensemble is successful for 96.3% of the proteins.

At the superfamily level, the ensemble classifier is not as close to the theoretical limit as it was for the family level. Yet, it still improves the performance of the individual tools. Performance improvements are between 1.4-2.9 times. the ensemble is successful for 86.1% of the queries. In Figure 7.4, the overlap between the correctly-identified proteins by the tools is minimum at the fold level. Figure 7.10 shows that the ensemble classifier outperforms individual tools 1.1-1.4 times by being successful for 89% of the queries.

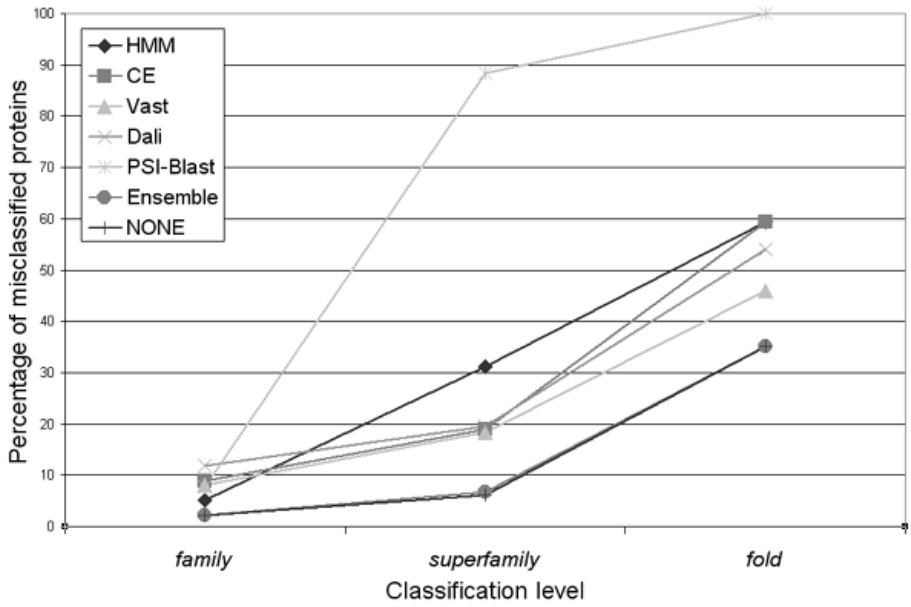

Figure 7.11: Performance of individual classifiers compared to the ensemble classifier on category assignment problem for the new proteins introduced in SCOP v1.61.

	Training	**Evaluation**
Database	$DS159$ (20449)	$DS161$ (22724)
Query	$QS161$ (2241)	$QS163$ (2825)
$newFam$	248	618
$newSF$	84	424
$newFold$	47	339

Table 7.2: Database and query data sets and their sizes.

The second part of training is the assignment of an existing class label to the new protein. Figure 7.10 shows that the ensemble classifier outperforms all the component tools. At the family level, none of the tools is successful for 2.1% of the queries, as can be seen in Figure 7.5. The ensemble classifier performs at this theoretical limit (97.9% of the proteins). The improvement at the family level is between 2.4-5.6 times. At the superfamily level, the ensemble classifier performs almost at the theoretical limit. It has an error rate of 6.7% whereas the theoretical limit is 6.1%. As a result, the performance of the tools has been improved 2.7-13 times. At the fold level, the ensemble has an error rate of 35% which is the same as the theoretical limit. Performance of the individual tools has been improved 1.3-2.8 times, and 65% of the proteins are assigned to the correct folds.

7.5.2 Validation Procedure

When we test our algorithm by using queries from $QS163$ on the database $DS161$, we see that the ensemble improves the performance as in Figure 7.10. None of the tools is able to perform better than ensemble on all levels. Although HMM

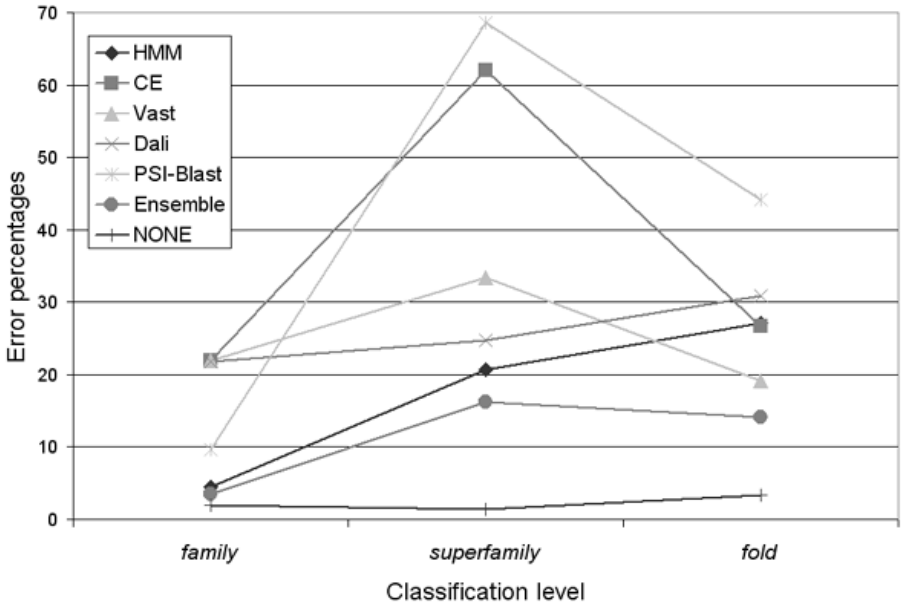

Figure 7.12: Performance of individual classifiers compared to the ensemble on category membership problem for the new proteins introduced in SCOP v1.63.

performs close to the ensemble classifier at the family level, it fails to compete at the superfamily and the fold levels. The ensemble outperforms HMM by 1.3 times at the family and superfamily levels. At the fold level the ensemble outperforms the best tool, Vast, by 1.4 times. The ensemble also manages high accuracies at all levels; 96.5% for family, 83.8% for superfamily, and 86% for fold levels.

After training the ensemble on the $DS159$ and $QS161$, we test it with the trained parameters on $DS161$ and $QS163$. By comparing Figure 7.11 and Figure 7.13, we can see the effect of improvement. We trained the ensemble classifier to perform close to the theoretical limit. When we tested it, it performs slightly worse than the theoretical limit, but significantly better than the individual tools. The improvement

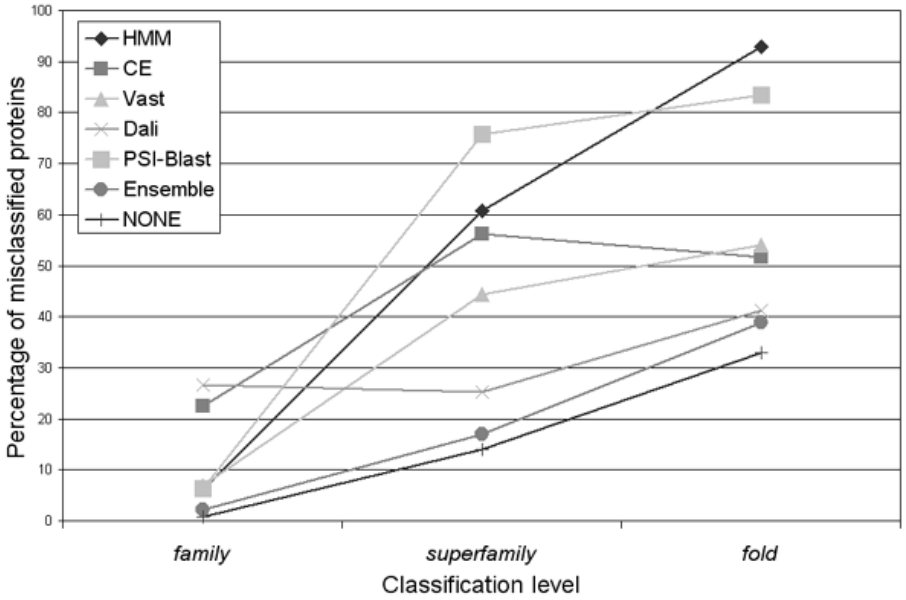

Figure 7.13: Performance of individual classifiers compared to the ensemble on category assignment problem for the new proteins introduced in SCOP v1.63.

at the family level is 3-12 times, at the superfamily level 1.5-4.5 times, and at the fold level 1.1-2.4 times. The ensemble classifier assigns 97.9% of proteins to correct families, and 83% of the proteins to the correct superfamilies, and 61.2% of the queries to the correct folds.

7.5.3 Error analysis

We have analyzed the query proteins where the ensemble fails to find the correct classifications. Most of these errors are due to factors that depend on human judgement rather than on computational results using sequence and structural similarity. Below, we present a few of such cases.

The protein structure 1kuu-A, an α+β class protein, is predicted by the ensemble to be a member of an existing superfamily *N-terminal nucleophile aminohydrolases*. It is very similar to the members of its predicted superfamily, according to both sequence and structure classifiers. But, it is actually a member of the *Hypothetical protein MTH1020* superfamily. The only difference between these two superfamilies is that the latter lacks the N-terminal nucleophile. Clearly SCOP authors decided to create a new superfamily based on information that cannot be inferred by automated classifiers. The false hits between these two superfamilies contribute to significant portion of the mistakes of the ensemble classifier for superfamily assignments.

For proteins in the *5-bladed beta-propeller* fold, classifiers get false hits from other structurally similar folds. CE and Dali assign proteins from this fold to the *6-bladed beta-propeller* fold, and Vast assigns them to the *4-bladed beta-propeller* fold. Since these folds have a similar layout, their members are prone to get short but high scoring alignments across classes. This is an example of the Russian Doll effect [68].

7.6 Discussion

The most trusted protein classification databases are the manually-curated ones. However, as new protein structures are continuously being discovered, there is a need to automatically update protein classification databases *timely* and *accurately* to account for the new structures. In this chapter, we explored the applicability of automated classification, and proposed a novel method that uses the consensus deci-

sion principle to obtain higher quality classifications of manually curated databases using automated techniques. Our technique significantly outperforms the individual component classifiers by achieving error rates that are 3-12 times less than the individual classifiers' error rates at the family level, 1.5-4.5 times less at the superfamily level, and 1.1-2.4 times less at the fold level. We envision that our technique can help researchers classify proteins in a completely automated manner. Even for manual classification, it can provide strong clues that will reduce the workload.

Chapter 8

Conclusions and Future Work

In this dissertation, we have presented methods that we have developed that can scale well with the increase in the amount of available structure data and help biologists analyze large numbers of protein structure data more efficiently. Our contribution can be described in three main categories: (1) visualization and surface modelling, (2) structure comparison and similarity search, and (3) automated classification.

1. For efficiently visualizing protein structures using a scene-graph based graphics API, we have presented methods to optimize the constructed scene-graph to enable real-time visualization of very large protein complexes. Our method (FPV) achieves up to 8 times interactive speed compared to existing methods. For molecular surfaces we have presented a method based on a level set formulation that can compute the surface and interior inaccessible cavities very efficiently (1.5 to 3.14 times faster on the average than compared methods).

2. For comparison and similarity search of protein structures we have presented a method that utilizes local shape signatures based on the theory of differential geometry. Our method (CTSS) is up to 30 times faster than CE in conducting protein structure similarity search in a large database of protein structures, while achieving the similar level of accuracy. We have also presented an integrated sequence and structure analysis method (ProGreSS), which enables biologists to perform joint sequence and structure similarity queries while improving on the accuracy and efficiency of existing methods.

3. For classification of protein structures automatically, we have presented an ensemble classifier framework based on decision trees rooted in machine learning. We have demonstrated that higher classification accuracy can be achieved using the joint hypothesis of the ensemble classifier.

The methods developed in this dissertation can be extended in the following ways:

1. For our solution for the molecular surface computation, the next step is to find the molecular surface dynamically as the probe radius changes. Using distance-transform one can generate a list of initial surface points (zero distance) and then propagate from those points outward to a particular distance, k (the maximum probe radius). The solvent-accessible surface for a particular probe radius $r < k$ is then readily computed by this method. The solvent-excluded surface can also be determined dynamically by shrinking the accessible surface using another distance transform (fixed-speed level set formula-

tion). All the points that are r distance from the initial surface will give the solvent-excluded surface.

2. Molecular surface comparison is a more difficult computational challenge compared to sequence or structure comparison methods. Finding surface similarities among a family of proteins may help reveal the functional determinant of that family, and the other dual problem of finding a complementary surface (the docking problem) may help in drug discovery and development. Several surface properties can be considered when comparing protein surfaces: electrostatic potential, surface curvature, cavity size, molecular surface area, and molecular volume.

3. In Chapter 4, we have developed a method for performing structural similarity queries using a geometric hashing based index structure. The index structure is also suitable for performing multiple structural alignments. Given a family of proteins as input, we can use the index to detect the most conserved local regions, as they will be accumulated into the same or close-by hash bins. The remaining task is to build the alignment as a consistent chain of local fragments.

Bibliography

[1] S. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[2] S. F. Altschul and E. V. Koonin. Iterated profile searches with PSI-BLAST–a tool for discovery in protein databases. *Trends Biochem Sci.*, 23(11):444–7, 1998.

[3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, W. Miller, and D. J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[4] J. An, T. Nakama, Y. Kubota, H. Wako, and A. Sarai. Construction of an integrated environment for sequence, structure, property and function analysis of proteins. *Genome Informatics*, 10:229–230, 1999.

[5] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 9(5):698–700, 1987.

[6] Z. Aung, W. Fu, and K. L. Tan. An efficient index-based protein structure database searching method. In *Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 311–318, Kyoto, Japan, 2003.

[7] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, B. A. Rapp, and D. L. Wheeler. Genbank. *Nucleic Acids Research*, 28:15–18, 2000.

[8] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[9] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is ”nearest neighbor” meaningful? In *Proceedings of 7th International Conference on Database Theory (ICDT)*, pages 217–235, Jerusalem, Israel, 1999.

[10] A. Bhattacharya, T. Can, T. Kahveci, A. K. Singh, and Y. F. Wang. Progress: Simultaneous searching of protein databases by sequence and structure. In *Proceedings of the 9th Pacific Symposium on Biocomputing (PSB)*, page Accepted, Big Island, HI, January 2004.

[11] T. L. Blundell and M. S. Johnson. Catching a common fold. *Protein Science*, 2(6):877–883, 1993.

[12] B. Boeckmann, A. Bairoch, R. Apweiler, M. C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O’Donovan, I. Phan, S. Pilbout,

and M. Schneider. The SWISS–PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31:365–370, 2003.

[13] S. E. Brenner, P. Koehl, and M. Levitt. The astral compendium for sequence and structure analysis. *Nucleic Acids Research*, 28:254–256, 2000.

[14] S. K. Burley, S. C. Almo, J. B. Bonanno, M. Capel, M. R. Chance, T. Gaasterland, D. W. Lin, A. Sali, F. W. Studier, and S. Swaminathan. Structural genomics: beyond the human genome project. *Nature Genetics*, 23:151–157, 1999.

[15] O. Camoglu, T. Kahveci, and A. K. Singh. Towards index-based similarity search for protein structure databases. In *Proceedings of Computational Systems Bioinformatics (CSB)*, pages 148–158, Stanford, CA, Aug 2003.

[16] T. Can, Y. Wang, Y. F. Wang, and J. Su. FPV: Fast protein visualization using java 3d. *Bioinformatics*, 19(8):913–922, 2003.

[17] T. Can and Y. F. Wang. CTSS: A robust and efficient method for protein structure alignment based on local geometrical and biological features. In *Proceedings of Computational Systems Bioinformatics (CSB)*, pages 169–179, Stanford, CA, Aug 2003.

[18] M. L. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16:548–558, 1983.

[19] M. L. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221:709–713, 1983.

[20] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5(3):345–352, 1978.

[21] W. L. DeLano. *The PyMOL Molecular Graphics System User's Manual*. DeLano Scientific, San Carlos, CA, USA, 2002.

[22] M. P. do Carmo. *Differential geometry of curve and surfaces*. Prentice-Hall, New Jersey, 1976.

[23] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, second edition, 2001.

[24] S. R. Eddy. Profile hidden markov models. *Bioinformatics*, 14:755–763, 1998.

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

[26] I. Eidhammer, I. Jonassen, and W. R. Taylor. Structure comparison and structure patterns. *Journal of COmputational Biology*, 7(5):685–716, 2000.

[27] D. Fischer, A. Elofsson, D. Rice, and D. Eisenberg. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. In *Proceedings of the 1st Pacific Symposium on Biocomputing (PSB'96)*, pages 300–318, Hawaii, 1996.

[28] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics Principles and Practice*. Addison-Wesley, second edition, 1990. Reading.

[29] M. Gerstein. Integrative database analysis in structural genomics. *Nat. Struct. Biol.*, Suppl:960–3, 2000.

[30] M. Gerstein and M. Levitt. Comprehensive assessment of automatic structural alignment against a manual standard, the scop classification of proteins. *Protein Science*, 7:445–456, 1998.

[31] M. Gerstein, F. M. Richards, M. S. Chapman, and M. L. Connolly. *Protein surfaces and volumes: measurement and use*, volume F of *International Tables for Crystallography. Crystallography of Biological Molecules*, chapter 22.1, pages 531–545. Kluwer Academic Publishers, Dortrecht, Netherlands, 2001.

[32] G. Getz, M. Vendruscolo, D. Sachs, and E. Domany. Automated assignment of SCOP and CATH protein structure classifications from FSSP scores. *Proteins*, 46:405–415, 2002.

[33] J. F. Gibrat, T. Madej, and S. H. Bryant. Surprising similarities in structure comparison. *Current Opinion in Structural Biology*, 6(3):377–385, 1996.

[34] W. Gish and D. J. States. Identification of protein coding regions by database similarity search. *Nature Genetics*, 3(3):266–272, 1993.

[35] A. Godzik. The structural alignment between two proteins: Is there a unique answer? *Protein Science*, 5(7):1325–1338, 1996.

[36] J. Gough. The SUPERFAMILY database in structural genomics. *Acta Cryst.*, D58:1897–1900, 2002.

[37] N. Guex and M. C. Peitsch. SWISS–MODEL and the Swiss–PdbViewer: An environment for comparative protein modeling. *Electrophoresis*, 18:2714–2723, 1997.

[38] A. Guéziec and N. Ayache. Smoothing and matching of 3-d space curves. *International Journal of Computer Vision*, 12(1):79–104, 1994.

[39] H. Hegyi and M. Gerstein. The relationship between protein structure and function: a comprehensive survey with application to the yeast genome. *Journal of Molecular Biology*, 288(1):147–164, 1999.

[40] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. National Academia Science*, 89:10915–10919, 1992.

[41] U. Hobohm, M. Scharf, R. Schneider, and C. Sander. Selection of a representative protein data sets. *Protein Science*, 1(3):409–417, 1992.

[42] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233(1):123–138, 1993.

[43] L. Holm and C. Sander. 3-D lookup: Fast protein structure database searches at 90% reliability. In *Proceedings of the 3rd International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 179–187, Cambridge, UK, 1995.

[44] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–602, 1996.

[45] C. C. Huang, G. S. Couch, E. F. Pettersen, and T. E. Ferrin. Chimera: an extensible molecular modeling application constructed using standard components. In *Proceedings of the Pacific Symposium on Biocomputing (PSB)*, volume 1, page 724, 1996.

[46] C. C. Huang, W. R. Novak, P. C. Babbitt, A. I. Jewett, T. E. Ferrin, and T. E. Klein. Integrated tools for structural and sequence alignment and analysis. In *Proceedings of Pasific Symposium in Biocomputing (PSB)*, pages 227–238, 2000.

[47] S. J. Hubbard and P. Argos. Cavities and packing at protein interfaces. *Protein Science*, Vol 3(12):2194–2206, December 1994.

[48] W. Kabsch and C. Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22:2577–2637, 1983.

[49] E. Kishon, T. Hastie, and H. Wolfson. 3-D curve matching using splines. *Journal of Robotic Systems*, 8(6):723–743, 1991.

[50] R. Koradi, M. Billeter, and K. Wüthrich. MOLMOL: a program for display and analysis of macromolecular structures. *Journal of Molecular Graphics*, 14:51–55, 1996.

[51] P. J. Kraulis. MOLSCRIPT: A program to produce both detailed and schematic plots of protein structures. *Journal of Applied Crystallography*, 24:946–950, 1991.

[52] Y. Lamdan and H. J. Wolfson. Geometric hashing: a general and efficient model-based recognition scheme. In *Proceedings of the 2nd International Conference on Computer Vision (ICCV)*, pages 238–249, Florida, US, 1988.

[53] R. H. Lathrop. The protein threading problem with sequence amino acid interaction preferences is np-complete. *Protein Engineering*, 7(9):1059–1068, 1994.

[54] N. Leibowitz, Z. Y. Fligelman, R. Nussinov, and H. J. Wolfson. Multiple structural alignment and core detection by geometric hashing. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 169–177, Heidelberg, Germany, 1999.

[55] 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, October 1998.

[56] E. Lindahl and A. Eloffson. Identification of related proteins on family, superfamily and fold level. *J. Mol. Biol*, 295:613–625, 2000.

[57] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[58] G. Lu. Top: a new method for protein structure comparison and similarity searches. *Journal of Applied Crystallography*, 33(1):176–183, 2000.

[59] J. Lundstrom, L. Rychlewski, J. Bujnicki, and A. Elofsson. Pcons: A neural-

network-based consensus predictor that improves fold recognition. *Prot. Sci*, 10:2354–2362, 2001.

[60] T. Madej, J. F. Gibrat, and S. H. Bryant. Threading a database of protein cores. *Proteins*, 23:356–369, 1995.

[61] R. Meir and G. Ratsch. *Advanced Lectures on Machine Learning*, chapter An introduction to boosting and leveraging. Springer Verlag, 2003.

[62] S. Meloan. Exploring the new frontier: Java technology powers the post-genomic era. *Feature Stories, java.sun.com*, September 2001.

[63] K. Mizuguchi, C. M. Deane, T. L. Blundell, M. S. Johnson, and J. P. Overington. Joy: protein sequence-structure representation and analysis. *Bioinformatics*, 14:617–623, 1998.

[64] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

[65] A. Nicholls. *GRASP: graphical representation and analysis of surface properties*. Columbia University, New York, 1992.

[66] A. Nicholls, K. Sharp, and B. Honig. Protein folding and association: insights from the interfacial and thermodynamic properties of hydrocarbons. *Proteins*, 11(4):281–296, 1991.

[67] R. Nussinov and H. J. Wolfson. Efficient detection of three-dimensional structural motifs in biological macromolecules by computer vision techniques. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 88(23):10495–10499, 1991.

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

[69] S. Osher and R. Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153 of *Applied Mathematical Sciences*. Springer-Verlag, 2003.

[70] X. Pennec and N. Ayache. A geometric algorithm to find small but highly similar 3d substructures in proteins. *Bioinformatics*, 14(6):516–522, 1998.

[71] E. Portugaly and M. Linial. Estimating the probability for a protein to have a new fold: A statistical computational model. *Proc. Natl. Acad. Sci.*, 97(10):5161–5166, May 2000.

[72] S. Rackovsky and H. A. Scheraga. Differential geometry and polymer conformation. 1. comparison of protein conformations. *Macromolecules*, 11:1168–1174, 1978.

[73] M. F. Sanner. Python: a programming language for software integration and development. *Journal of Molecular Graphics and Modelling*, 17(1):57–61, February 1999.

[74] M. F. Sanner, A. J. Olson, and J. C. Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996.

[75] M. A. S. Saqi, D. L. Wild, and M. J. Hartshorn. Protein analyst - a distributed object environment for protein sequence and structure analysis. *Bioinformatics*, 15:521–522, 1999.

[76] J. M. Sauder, J. W. Arthur, and R. L. D. Jr. Large-scale comparison of protein sequence alignment algorithms with structure alignments. *Proteins: Structure, Function, and Genetics*, 40(1):6–22, 2000.

[77] R. Sayle and E. J. Milner-White. RasMol: Biomolecular graphics for all. *Trends in Biochemical Sciences (TIBS)*, 20(9):374–376, September 1995.

[78] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[79] J. A. Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision and materials science*. Cambridge University Press, 2nd edition, August 1999.

[80] I. N. Shindyalov and P. E. Bourne. Protein structure alignment by incremental combinatorial extension (ce) of the optimal path. *Protein Engineering*, 11(9):739–747, 1998.

[81] I. N. Shindyalov and P. E. Bourne. An alternative view of the protein fold space. *Proteins*, 38:247–260, 2000.

[82] C. Shirky. Seven ways of looking at a protein. *FEED Magazine*, page After Darwin Column, October 2000.

[83] A. P. Singh and D. L. Brutlag. Hierarchical protein structure superposition using both secondary structure and atomic representations. In *Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 284–293, Halkidiki, Greece, 1997.

[84] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[85] H. Sowizral, K. Rushforth, and M. Deering. *Java 3D API Specification (Version 1.1.2)*, June 1999.

[86] K. Takano, Y. Yamagata, and K. Yutani. Buried water molecules contribute to the conformational stability of a protein. *Protein Engineering*, 16(1):5–9, January 2003.

[87] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Vision (PAMI)*, 13(4):376–380, 1991.

[88] D. Walther. WebMol - a java based PDB viewer. *Trends in Biochemical Sciences*, 22:274–275, July 1997.

[89] H. J. Wolfson and I. Rigoutsos. Geometric hashing: An introduction. *IEEE Computational Science & Engineering*, 4(4):10–21, 1997.

[90] T. C. Wood and W. R. Pearson. Evolution of protein sequences and structures. *Journal of Molecular Biology*, 291(4):977–995, 1999.

[91] M. M. Young, A. G. Skillman, and I. D. Kuntz. A rapid method for exploring the protein structure universe. *Proteins: Structure, Function, and Genetics*, 34(3):317–332, 1999.