INDEXING METHODS FOR PROTEIN TERTIARY AND PREDICTED STRUCTURES

By

Feng Gao

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the
Examining Committee:

_____
Mohammed Zaki, Thesis Adviser

_____
Christopher Bystroff, Member

_____
Sibel Adalı, Member

_____
David Musser, Member

Rensselaer Polytechnic Institute
Troy, New York

December 2006
(For Graduation December 2006)

INDEXING METHODS FOR PROTEIN TERTIARY AND PREDICTED STRUCTURES

By

Feng Gao

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Mohammed Zaki, Thesis Adviser

Christopher Bystroff, Member

Sibel Adalı, Member

David Musser, Member

Rensselaer Polytechnic Institute
Troy, New York

December 2006
(For Graduation December 2006)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to start by thanking my advisor Prof. Mohammed Zaki. His vision and enthusiasm have made my education a rich experience and I could not have achieved my academic success without his guidance. Under his supervision, not only do I learn how to solve problems, but also learn how to solve them in an efficient manner.

Special thanks go to Prof. Chris Bystroff for very helpful discussion which initiated the idea of predicted structure indexing. I also want to thank Prof. Sibel Adali and David Musser for spending their valuable time as my thesis committee members and providing insightful comments.

My current and previous group members broaden my views through the exchange of research ideas. I would like to thank Benjarath Phoophakdee, Yongqiang Zhang, Lizhuang Zhao, Zujun Shentu, Vineet Chaoji, Saeed Salem, Mohammad Al Hasan.

Finally, I would like to thank my parents, my sister and my brothers, who made me all I am today. Although they live in other places, they have always been there for me to provide me with support. And most importantly I must thank my wife, Xubei, who not only supports me spiritually, but also takes care of my life. Without her love and encouragement I could not have made this far. I dedicate this thesis to my parents and my wife.

ABSTRACT

This thesis focuses on the problem of fast sub-structure search and remote homology detection in proteins by finding similar (sub)structures. That is, for a given query protein and a large database of protein structures, we want to retrieve all the similar structures from the database rapidly. With the growing number of proteins deposited in the database, searching the database is a difficult and time-consuming task. In fact, high throughput proteomics methods are already accumulating the protein interaction data that we would wish to model, but fast computational methods for database searching lag far behind; biologists are in need of a means to search the protein structure databases rapidly, similar to the way BLAST rapidly searches the sequence databases.

We are interested in two main problems that arise in sub-structure and remote homology searches, namely protein tertiary structure indexing and predicted structure indexing for those proteins whose structures have not been determined experimentally. In our tertiary structure indexing approach, a new method for extracting the local feature vectors of protein structures is presented. Each residue is represented by a triangle, and the correlation between a set of residues is described by the distances between C_α atoms and the angles between the normals of planes in which the triangles lie. The normalized local feature vectors are indexed using a suffix tree. For all query segments, suffix trees can be used effectively to retrieve the maximal matches, which are then chained to obtain alignments with database proteins. Similar proteins are selected by their alignment score against the query. In our predicted structure indexing approach, a hidden Markov model (HMMSTR) of high sequence-structure local motifs (I-sites library) is used to generate the feature vectors for the structure predicted for a given sequence. Remote homologous proteins are detected by using the suffix tree index over the predicted structures. We test our algorithms on several real datasets. We improve both the time and accuracy perfor-

mance of the tertiary structure indexing and classification. We also find more remote homologous proteins from the database of predicted structures than competing methods.

CHAPTER 1

Introduction

1.1 Motivation

Proteins are basic building blocks of all organisms and are indeed fundamental to life, so understanding their biological structure and function is not only important for the study of biological processes, but also produces very valuable practical benefits such as in drug design and genetic engineering.

Proteins are composed of chains of amino acids, which determine the tertiary (or three-dimensional; 3D) structure of the protein via the process of protein folding. Breakthroughs in large-scale sequencing have led to a deluge of sequence information, and advances in experimental techniques such as NMR and X-ray crystallography in molecular biology have led to the determination of 3D structure of many proteins. However, the determination of protein function lags behind: there are still 40-50% of new proteins whose functions remain unknown.

A protein's function is determined by its native conformation including surface structure, binding sites and active sites plus the biochemical and biophysical properties of its constituent amino acids. Proteins that share a common ancestor are called homologous. Homologous proteins tend to have common tertiary structure and active sites, and frequently share common functions.

The function of a protein is determined in part from its sequence of amino acids; however, biologists have determined that even proteins which are remotely homologous in their sequential similarities can perform surprisingly very similar functions in living organisms [54]. This fact has been attributed to the dependency of the functional role of proteins to their actual 3D structure. Two or more structures are said to be homologous if

they are alike because of shared ancestry. In view of this it can be stated that two proteins with remote sequence homology can be functionally classified as similar if they exhibit structural homology.

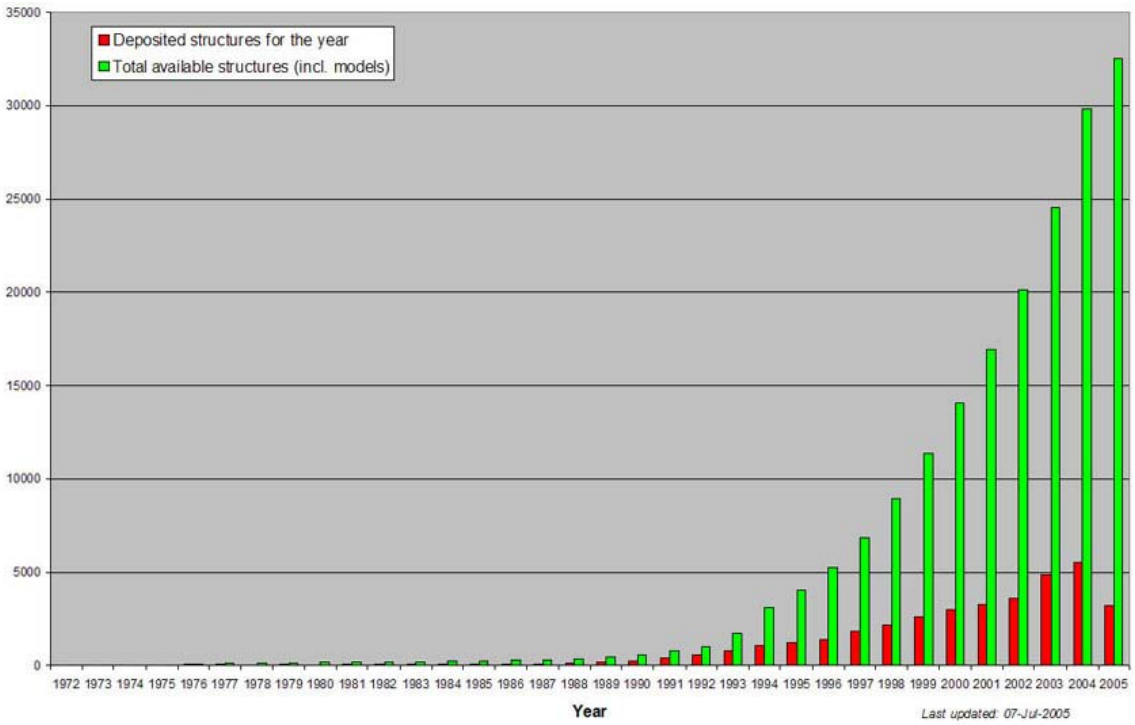

Figure 1.1: Exponential Growth of PDB (as of July 2005, www.pdb.org)

On the other hand, the database of known protein structures, Protein Data Bank (PDB) contains around 31,971 proteins, and new structures are being added to this database at the rate of 400-500 per month (see Figure 1.1). However even though scientists are able to determine structures at such a phenomenal rate, this data becomes relevant only if the functionalities of the proteins are also known. To do this we need to find the structurally homologous proteins of the newly found proteins by comparing their structures with previously known proteins whose structures and functionality are both known.

Our goal in this thesis is to be able to detect remote homologous proteins by finding similar structures from a large database for a given query protein. In other words, for a given query protein and a large structure database, we need to retrieve all the similar structures from the database rapidly. With the growing number of proteins deposited in

the database, searching the database is a difficult and time-consuming task. For example, we may want to retrieve all structures that contain sub-structures similar to the query, a specific 3D arrangement of surface residues, etc. Searches such as these are the first step towards building a systems level model for protein interactions. In fact, high throughput proteomics methods are already accumulating the protein interaction data that we would wish to model, but fast computational methods for database searching lag far behind; biologists are in need of a means to search the protein structure databases rapidly, similar to the way BLAST [1] rapidly searches the sequence databases.

1.2 Biological background

In this section, we present some background on protein structure. At first, we describe the structure of proteins at different levels, then discuss various methods to determine protein tertiary structures, including the experimental and prediction methods.

1.2.1 Protein structure

Proteins are amino acid chains that fold into unique 3-dimensional structures. The shape into which a protein naturally folds is known as its native state, which is determined by its sequence of amino acids [69]. Proteins have multiple level of structures: primary structure, secondary structure, tertiary structure and quaternary structure.

Protein primary structure refers to the linear sequence of amino acids (see figure 1.2). There are 20 common amino acids which have similar but unique structures. An amino acid has a central α-carbon atom to which it is attached. In addition is has an amino group,a carboxyl group, a hydrogen atom, and a unique side chain (R group). There are 20 different R groups corresponding to the 20 common amino acids. An amino acid residue is what is left of an amino acid once a molecule of water has been lost (an H+ from the amino group and an OH- from the carboxyl group) in the formation of a peptide bond, the chemical bond that links the amino acid monomers in a protein chain.

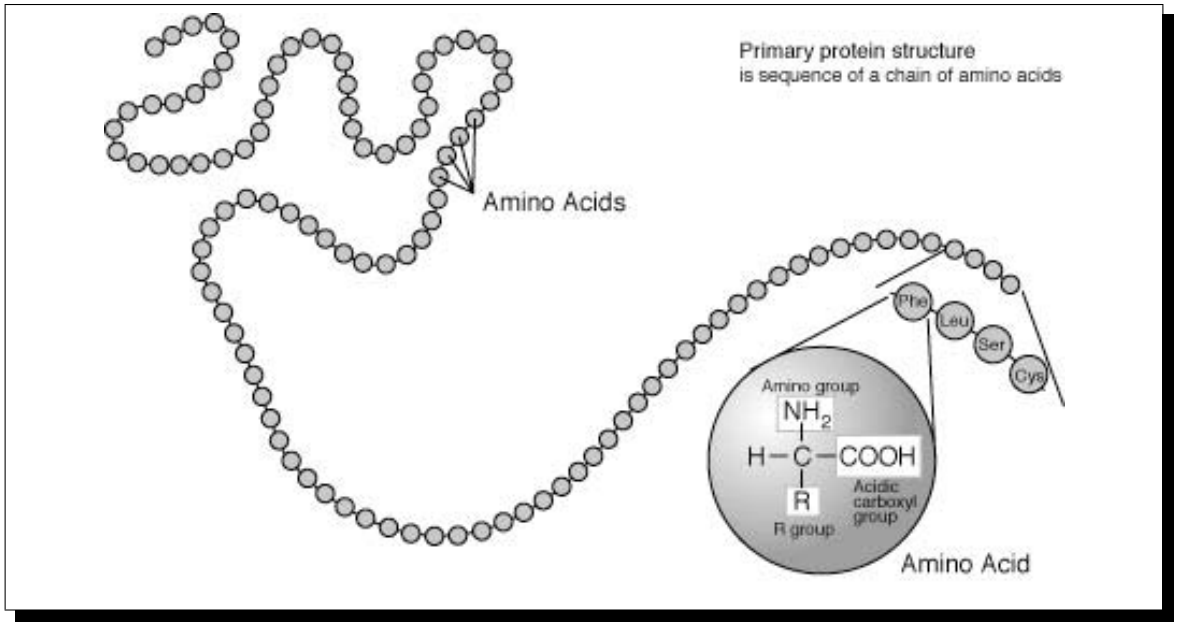

Figure 1.2: Protein primary structure and Amino Acids (en.wikipedia.org)

Protein secondary structure refers to the local ordered sub-structures with specific geometric shape. There are two common secondary structures: alpha helix and beta sheet. An alpha helix is characterized by a single, spiral chain of amino acids stabilized by hydrogen bonds, and a beta sheet consists of two or more amino acid sequences within the same protein that are arranged adjacently and in parallel, but with alternating orientation such that hydrogen bonds can form between the two strands. In figure 1.3, the red spirals represent alpha helices, and the green arrows the beta sheets.

Protein tertiary structure refers to the global folding of a single chain. Figure 1.3 gives an example of protein tertiary structure. Finally, the quaternary structure involves the association of two or more chains into a multi-subunit structure.

1.2.2 Protein structure determination

There are three methods to determine the structure of proteins: X-ray crystallography, Nuclear Magnetic Resonance (NMR) and structure prediction. The first two are experimental methods.

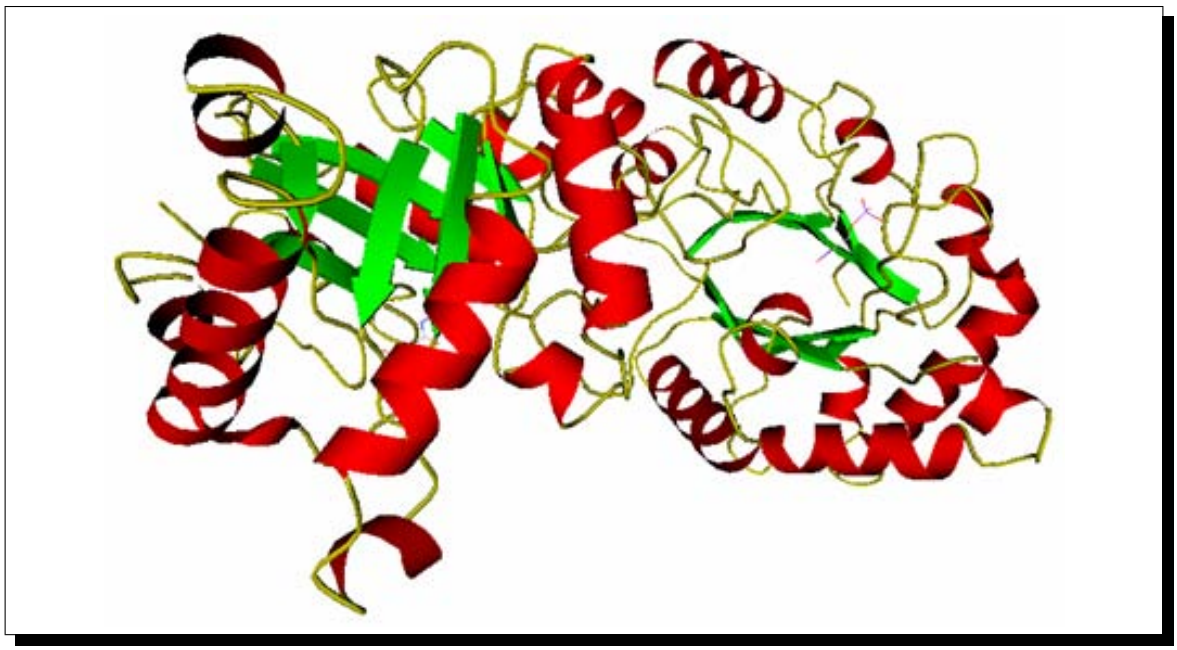

Figure 1.3: Protein secondary and tertiary structure

X-ray crystallography has been used to determine the structure of inorganic and organic crystals since the early years of this century. It is based on scattering from electrons and is suitable for any size of molecule. NMR obtains the same high resolution using a very different strategy. NMR measures the distances between atomic nuclei, rather than the electron density in a molecule. NMR determines structures of proteins in solution, but is limited by the molecule size. NMR is the method of choice for small proteins which are not readily crystallized, and yields the positions of some hydrogen atoms.

The goal of structure prediction is to determine the 3D structure of proteins from their amino acid sequences. There are two classes of prediction methods; ab initio modeling and comparative modeling. Protein ab initio modeling methods try to build 3D protein models ”from scratch,” while comparative modeling methods build a model of the novel protein based upon known structures.

1.3 Challenges in protein indexing

As mentioned in the section 1.1, we aim to detect homologous proteins by finding similar proteins from a large database for a given query protein. To search for similarity over huge number of proteins, database indexing and scalable searching approaches are required [2, 5, 9, 10, 1]. Depending on whether the protein structures have been determined or not, we present the challenges separately for the area of tertiary structure indexing and predicted structure indexing.

1.3.1 Tertiary structure indexing

If the protein tertiary structures have been determined experimentally, we can detect homologous proteins directly by structure similarity search. However, structure similarity comparison is an NP-hard problem [21], so there are no fast structure alignment algorithms that can guarantee optimality using a given similarity measure. Thus, current structure comparison approaches tend to be heuristics-based.

Various methods such as the pairwise and multiple structure alignment approaches [24, 59, 15, 58] have been introduced. However, they are not suitable for searching for similarity over thousands of protein structures due to their time complexity. For a given query protein and a large structure database, we need to retrieve all the similar structures from the database rapidly. Database indexing and scalable searching approaches satisfy this requirement [2, 5, 9, 10].

The problem of finding similar proteins from the large database is especially challenging due to a number of reasons:

- Feature representation: Each atom is a point in the 3D space, and its coordinates depend on the origin and axis. The relationship between atoms in 3D space include both the translational and rotational degree. Thus, we need a precise feature representation.

- Similarity measurement: The similarity of proteins is usually defined by the score accumulated from their residues. For a given query, one of the most common similarity scoring schemes is the number of votes accumulated from the matching residues [5, 10, 33]. Another scoring scheme is the p-value of a protein based on the number of votes, where smaller p-values imply better similarity [10, 5]. However, current methods assume that the individual feature vectors are independent from each other, and the relationships of the components such as the order information and number of gaps between continuous residues are ignored. Thus, these scoring schemes are point-based and do not take into account the local segment similarity.

- Complexity: Currently, indexing methods such as R^*-trees and hash indexing can only retrieve the match points/features for a single feature vector of the query. Thus, the matching results might include a lot of random hits which do not contribute to the alignment between the query and template proteins from the database. So we need to retrieve only the significant matching feature vectors to reduce the search complexity.

1.3.2 Predicted structure indexing

Large scale sequencing has led to a surge of protein sequence information. For instance, swissprot [63] has more than 188,477 protein sequences. However, the process of determining the 3D structure of the proteins experimentally is still very time consuming. Thus, the output of experimentally determined protein structures is lagging far behind the output of protein sequences. The database of known protein structures, Protein Data Bank (PDB) only contains around 31,971 proteins. Therefore, for those proteins whose tertiary structures are unknown, we need to detect homologous proteins via sequence comparison.

Traditionally the problem of homolog detection was approached by finding proteins with high sequence similarity [61, 45, 1]. However, distantly related sequences share

fewer identical residues and less similarity, and remote homologous proteins can share a common structure and function even when they have insignificant sequence similarity. It is very difficult to detect those homologous proteins that fall in the twilight zone of 20-35% sequence identity if only the sequence information is used, so the other information such as structure must be considered.

Remote homology detection approaches [26] focus on detecting structurally homologous proteins with little sequence similarity. The structure information used in remote homolog detection methods is predicted from the protein amino acid sequence. Although the tertiary structure of proteins might be unknown, many structure features such as second structure, backbone angle and structural context descriptors [8] can be predicted by computational or experimental techniques quickly. Thus, structure prediction provides valuable information for the large fraction of sequences whose structures have not been determined experimentally.

Currently, most remote homolog detection methods [26, 68] align two predicted structures pairwise, but they are time-consuming. On the other hand, protein sequence indexing methods such as PSI-BLAST [1] have been widely used for homolog detection, but they rely on the sequence information and are not able to detect remote homologous proteins. Taken together, our second challenge is to overcome the limits of sequence homology detection and slow structure-based comparison and searching.

1.4 Contributions

Having highlighted the challenges in the area of protein tertiary structure indexing and predicted structure indexing, we now turn to the contributions of this thesis. For proteins with known structure, we index the structures directly, whereas for proteins with unknown structures we construct new structural features via prediction, and then index them.

1.4.1 Indexing tertiary structures

For known structures, we present a fast, novel protein indexing method called PSIST (which stands for **P**rotein **S**tructure **I**ndexing using **S**uffix **T**rees) in chapter 3. As the name implies, our new approach transforms the local structural information of a protein into a "sequence" on which a suffix tree is built for fast matches.

We first extract the feature vectors, then normalize and discretize the feature vectors, and the protein structure is converted into a sequence, called the *structure-feature sequence* or *SF-sequence*, of discretized symbols. We use suffix trees to index the protein SF-sequences. For a given query, all the maximal matches are retrieved from the suffix tree and chained into alignment, and the top proteins with the highest alignment scores are finally selected. Given two sequences, a maximal match is a matching subsequence that is longer than a threshold and is not contained in any such matching subsequence.

The main ideas behind PSIST are as follows: 1) To improve the speed and save space, PSIST converts the 3D structure indexing problem into a 2D sequence indexing problem. 2) All the residues are treated independently and equally in most current methods. However, PSIST encodes the order information of the residues for each protein structure, and only the continuous matching residues or matching segments are considered. Compared with a set of matching residues, a sequence of matching residues models the protein more closely. 3) Suffix tree is a versatile and fast data structure for substring problems [23], so it is chosen to index all the protein sequences and search all the maximal matching subsequences.

More specifically, we aim to counter the challenges mentioned in section 1.3.1 as follows:

- Feature representation: We extract local structural feature vectors using a sliding window along the backbone. For a pair of residues, the distance between their C_α atoms and the angle between the planes formed by the C_α, N and C atoms of

each residue are calculated. The feature vectors for a given window include all the distances and angles between the first residue and the rest of the residues within the window. Compared with the local features from a single residue, our feature vectors contain both the translational and rotational information.

- Similarity measurement: We define a new similarity measure between the query and the template protein based on the longest chains of maximal matching segment. Compared with other residue-based similarity measures, chained matching segment measurement encodes the order information and locality of the adjacent matching residues. A chain is obtained by chaining the maximal matches and its score is the sum of the score of the maximal matches minus the gaps penalty. The similarity score between the query and the template is defined as the maximum score of the chains. In our similarity scoring scheme, both the position and length of the maximal matches in the protein backbone and the penalty of the gaps between the segments are taken into account.

- Complexity: We use a suffix tree to index the protein sequence and retrieve the matching segments from the suffix tree. If a matching feature vector does not belong to any significant segments, it will not be retrieved from the suffix tree. Thus, the number of matching feature vectors is reduced. For instance, searching the whole query sequence from the suffix tree is equivalent to the exact sequence match [67], which has complexity $O(m|\Sigma|)$, where m is the length of the query and $|\Sigma|$ is the alphabet size of protein sequences. Please refer to section 3.2.1 for more details.

1.4.2 Indexing predicted structures

As discussed in section 1.3.2, purely sequence-based algorithms cannot detect remotely homologous proteins, and current structure-based algorithms are very slow. To overcome the limits of sequence homology detection and to provide a more powerful

database search method, we combine structure prediction techniques with structure indexing methods and present a powerful predicted structure search method.

For each protein with unknown tertiary structure, its local structure can be predicted by the HMMSTR approach [8] from the sequence. HMMSTR converts a protein sequence to a γ matrix, where each element is the probability that each residue at a given position is associated with one of the Markov states. After clustering, the γ matrix is converted to a sequence of distance vectors using two different methods: k-means and three secondary structural states feature. We then use a suffix tree to index all the sequences of distance vectors in the dataset. For each protein sequence, its homologous proteins are retrieved by searching the suffix tree.

1.5 Results

For PSIST, we perform three tests on both in-memory and external suffix trees: retrieval test, classification test, and time performance test. The retrieval test finds the number of correct matching structures from the same superfamily as the query among the top k scoring proteins. The classification test tries to classify the query at the superfamily and class level. The time performance test compares the running time of different algorithms.

For in-memory suffix tree tests, our results show classification accuracy up to 97.8% and 99.4% at the superfamily and class level according to the SCOP classification, and shows that on average 7.49 out of 10 proteins from the same superfamily are obtained among the top 10 matches. These results are competitive with the best previous methods. Compared to ProGreSS, our approach either obtains higher accuracy, or runs faster with similar classification accuracy.

For external suffix tree tests, we choose a dataset with low sequence similarities: all the sequence pairs have at most 35% residue sequence identity. The results show that our

algorithm is much better than ProGreSS. For instance, ProGreSS uses both the structure and sequence features to classify the proteins, and its accuracy is 7.14% and 57.1% at the topology and class levels. Without considering the sequence features, PSIST has much better performance than ProGreSS; its accuracy is 50.0% and 92.9% at the topology and class levels.

We also obtained results for predicted structure indexing. For a given query, its top k scoring homologous proteins are selected from the database. If a query and its best scoring homologous protein have the same topology, the homologous protein is considered a true hit. Our algorithms return 33-34 true hits while PSI-BLAST finds only 22. Receiver Operating Characteristic (ROC) [11] score is also used to measure the algorithms' performance, and our algorithm obtains a higher ROC score.

1.6 Outline

The rest of the thesis is organized as follows. Chapter 2 discusses the related work on structure indexing, homolog detection and application of suffix trees in protein sequence indexing. Chapter 3 presents our novel method to index the 3D structures using suffix trees. Chapter 4 develops a new method to index the structures predicted from sequences, and chapter 5 discusses future work.

CHAPTER 2

Related Work

In this chapter, we describe related work. At first, structure similarity search based on tertiary structure is described, then homolog detection methods based on both sequence and structure information are discussed. We also present the application of suffix trees in sequences indexing, genome alignment and structural motifs.

2.1 Structure similarity search

Protein structural similarity determination can be classified into three approaches: pairwise alignment, multiple structure alignments, and database indexing. Compared with pairwise alignment and multiple structure alignments, database indexing is less accurate but much faster. Thus, database indexing is suitable for searching similarity from a large number of proteins.

2.1.1 Protein structure alignment

Pair-wise structure alignment methods can be classified into three classes [18]. The first class works at the residue level [24, 59]. The second class focuses on using Secondary Structure Elements (SSEs) such as α-helices and β-strands to align two proteins approximately [35, 38, 43]. The third approach is to use geometric hashing, which can be applied at both the residue [33] and SSE level [25]. Geometric hashing has $O(n^3)$ complexity, where n is the number of elements (residues or SSEs). Pennec [46] shows how to reduce the complexity of geometric hashing to $O(n^2)$ time, using a hash table to index the transformation between two reference frames attached to each residue, where reference frame is defined as a set of coordinate axes in terms of which position or movement may be specified or with reference. For a given query structure, most of the pairwise struc-

ture comparison algorithms need to do an exhaustive sequential scan through the entire database to find matching structures.

Previous work has also looked at multiple structure alignments. These methods are also based on geometric hashing [41], or SSE information [15]. Multiprot [58] aims to solve the multiple structural alignment problem with detection of partial solutions; it computes the best scoring structural alignments, which can be either sequential or sequence-order independent [72], if one seeks geometric patterns which do not follow the sequence order.

2.1.2 Database structure indexing

Database structure indexing mainly contains two phases: In the off-line *indexing phase*, for each protein in the database, a set of local feature vectors are extracted and indexed using some spatial/metric indexing technique. The on-line *querying phase* involves three steps: for a query protein, its feature vectors are extracted, and then all the matching entries within a small distance range ϵ are retrieved from the database index. Using a suitable scoring scheme, the top k proteins with the highest similarity are then reported. Optionally, a multiple or pair-wise alignment between the query and matching proteins may be performed to report the minimum RMSD (root mean square deviation) values, if desired.

There are two classes of protein structure indexing approaches according to the representation of the local features. The first class focuses on indexing the local features at the residue level directly [10, 5, 11], and the other class uses SSEs to approximate the local feature of the proteins [9, 2].

CTSS [10] approximates the protein C_α backbone with a smooth spline with minimum curvature (see Figure 2.1). The method then stores the curvature, torsion angle and the secondary structure that each C_α atom in the backbone belongs to, in a hash-based index.

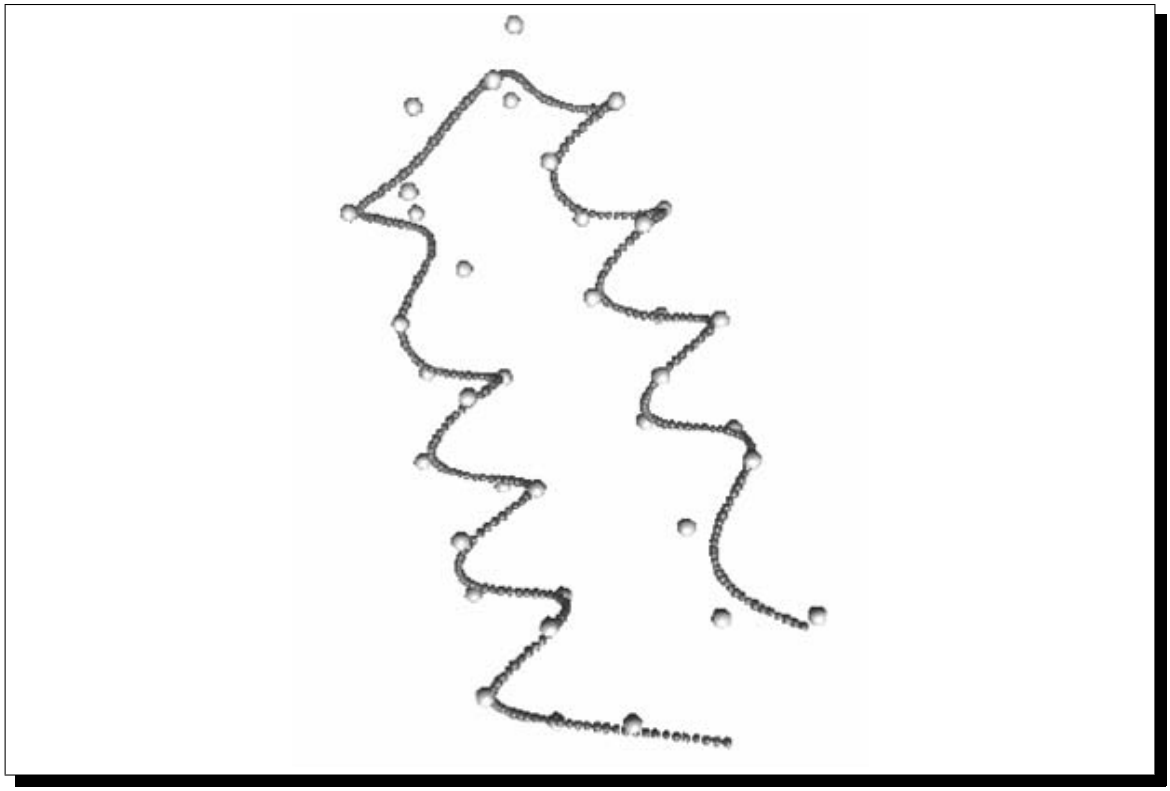

Figure 2.1: Spline approximation for C_α coordinates [10]

ProGreSS [5] is a recent method, which extracts the features for both the structure and sequence, within a sliding window over the backbone. Its structure features are the same as the CTSS features (curvature, torsion angles, and SSE information); its sequence features are derived using scoring matrices like PAM or BLOSUM. Like CTSS, ProGreSS features are not localized.

The LFF profile algorithm [11] first extracts some representative local features from the distance matrix of all the proteins, and then each distance matrix is encoded by the indices of the nearest representative features. Each structure is represented by a vector of the frequency of the representative local features. The structure similarity between two proteins is the Euclidean distance between their LLF profile vectors. This method is more suitable for global rather than local similarity between the query and database proteins.

There are also some methods that index the protein structures using SSEs. For each protein, PSI [9] uses a R^*-tree to index a nine-dimensional feature vector, a rep-

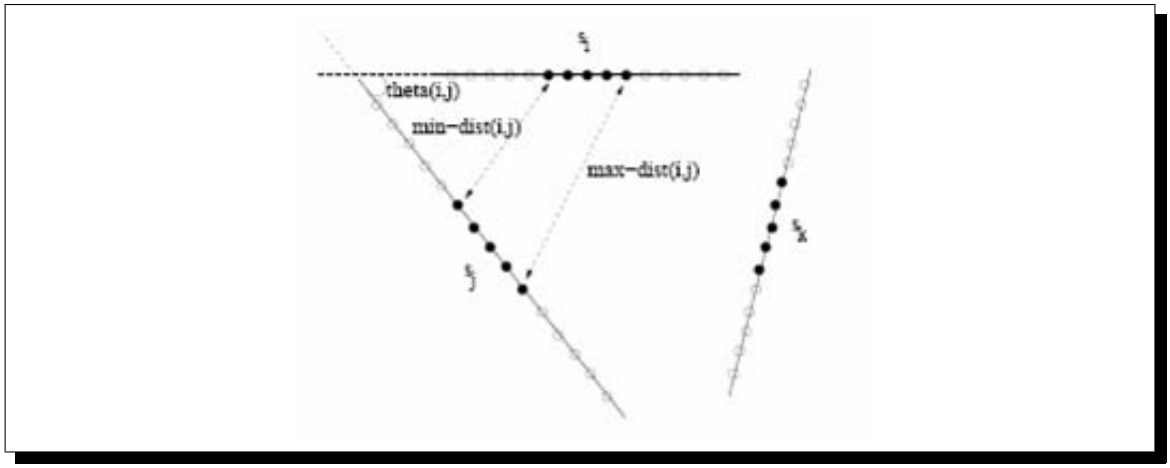

Figure 2.2: The feature vector for triplet $< s_i, s_j, s_k >$ [9]

resentation of all the triplet SSEs within a range. Figure 2.2 shows a triplet of SSEs $< s_i, s_j, s_k >$, and 3 feature vector, min-dist(i, j), max-dist(i, j) and theta(i, j), between s_i and s_j, where min-dist(i, j) is minimum distance between all pairs of black points from s_i and s_j, max-dist(i, j) is maximum distance between all pairs of black points from s_i and s_j, and theta(i, j) is the angle between the line segment approximations of s_i and s_j. Since each triplet consists of three pairs, the feature vector of each triplet contains 9 values. After retrieving the matching triplet pairs, a graph-based algorithm is used to compute the alignment of the matching SSE pairs.

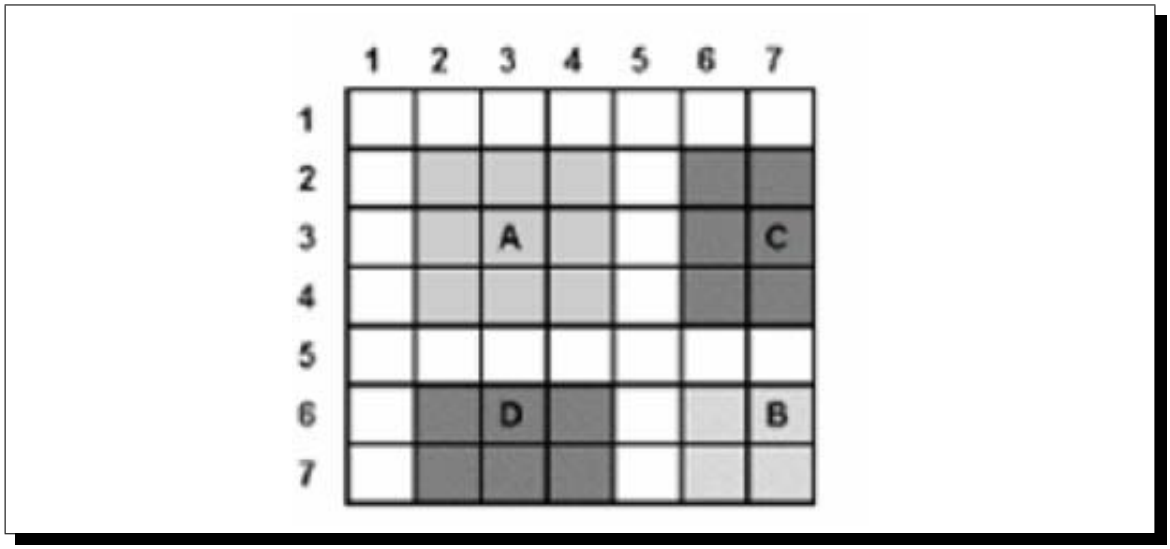

Figure 2.3: The SSE contact patterns [2]

Another SSE-based method, ProtDex [2] obtains the sub-matrices of the SSE contact patterns from the distance matrix of a protein structure. In the figure 2.3, the residue numbers 2 to 4 form a α-helix, and 6 to 7 form a β-sheet. The intersection of two SSEs forms a contact pattern and there are 4 contact patterns in the figure. The grand sum of the sub-matrices and the contact-pattern type are indexed by an inverted file index. By their nature, SSEs model the protein only approximately, and therefore these SSE-based approaches lack in retrieval accuracy and, furthermore, are not very useful for small query proteins with few SSEs.

ProteinDBS [60] is a recent algorithm that uses computational vision methods to convert the protein distance matrix into a gray-scale image, whose segmented portions are stored in an index.

By their nature, SSEs model the protein only approximately, and therefore these SSE-based approaches lack in retrieval accuracy and, furthermore, are not very useful for small query proteins with few SSEs.

2.2 Homology detection

There are two classes of homology detection approaches according to the information encoded in the feature vectors: sequence-based and sequence-structure-based, which contain both sequence and structure information. We distinguish three classes of sequence-based alignment approaches: sequence-sequence methods, profile-sequence methods and profile-profile methods.

Sequence-sequence methods compare the similarity of two primary sequences. Many algorithms have been developed such as dynamic programming[61], FASTA[45] and BLAST[1]. Dynamic programming is accurate but computationally expensive. FASTA and BLAST are two popular techniques to search a database rapidly for similar sequences approximately. However, sequence-sequence comparison algorithms cannot detect re-

mote homologous proteins which have insignificant sequence similarity, although they share a common fold and function.

The sequences from each family have the same evolutionary origin. Multiple alignments of remote homologous proteins provide more information about the family, indicating patterns of conservation at each position. A statistical model called a profile can be constructed by multiple alignment. A profile represents strongly conserved features throughout the family, thus, it allows prediction of similarity to a remote sequence (or superfamily), even if its similarity to each of the individual aligned sequences is insignificant.

There are two representative profile models: frequency profile and profile hidden Markov models (HMM). Frequency profile could be represented by a matrix of dimensions $20 \times L$, where L is the length of the target or query sequence. It contains information on the probability for the occurrence of each amino acid in each column of a multiple sequence alignment of proteins related to the query sequence.

Compared with simple frequency profiles, profile HMMs [16, 17] contain the position-specific probabilities for inserts and deletions along the multiple sequence alignment, in addition to the amino acid frequencies in the columns of the alignment. It has better performance ins homolog detection at the price of slower running times.

Well-known profile-sequence comparison approaches include PSI-BLAST [1], IMPALA [56] and HMMER [16]. For a given sequence, these methods will compare it with each family profile and compute how well the sequence fits the family profile.

Profile-profile comparison methods align two profiles, from which a similarity score and pairwise alignment of two sequences can be derived. Profile-profile comparison methods have identified some evolutionary links between protein families and led to a significant improvement over the profile-sequence methods. LAMA [48] compared a sequence alignment with a database of conserved ungapped alignments that characterize protein superfamilies. PROF_SIM [71] and COMPASS [55] allow for gaps and use the

Smith-Waterman local alignment algorithms. HHsearch [62] uses HMM-HMM comparison to detect homologous proteins.

Compared with profiles, Support Vector Machine (SVM) based approaches have better performance, because SVMs can model the difference between positive and negative examples (positive if they are in the same family and negative otherwise). An SVM algorithm chooses a hyperplane through the feature space that has a maximum margin between positive and negative examples. Each hyperplane divides one family from all other families. The similarity between two protein sequences is measured by the kernel function, a core component of a SVM. Many kernel functions have been developed and applied to remote homolog detection [30, 32].

Homologous proteins have a larger probability to have similar secondary structures than chance hits. Thus, several methods [68, 62] use predicted secondary structure for remote homolog detection and develop different substitution matrices for secondary structure states respectively. SVM-HMMSTR uses a predicted structure I-sites library for SVM training and homolog detection[26].

2.3 Suffix trees

A Suffix tree is a versatile data structure for substring problems [23]. It can be constructed in $O(n)$ time and space [36, 67]. Recently, suffix trees have been widely used for protein sequence database indexing [29, 37], genome alignment [13, 14] and extracting structural motifs.

To search for the similar sequences from a database accurately and efficiently, Meek [37] proposed an online and accurate technique for local-alignment searches on biological sequences, called OASIS. OASIS is a novel search algorithm which employs a dynamic programming search technique driven by traversing a suffix tree. The suffix tree is searched using a *best-first* (A^*) strategy. For a given query, an upper bound of matching

the remaining portion is computed during partial matching. OASIS always chooses the path that is capable of obtaining the highest scoring alignment in the process of searching, Thus, OASIS doesn't need to search the entire query and runs very fast.

To align two genomes, MUMer [13] first constructs a suffix tree, then retrieves all the maximal unique matches (MUMs) from the suffix tree. The longest increasing subsequence (LIS) and Smith-Waterman alignment are used further to obtain the final alignment. The novelty of MUMer is the integration of the above ideas.

To identify the structural motifs from the proteins, PAST [66] uses a coding of the backbone torsion angles and builds a generalized suffix tree for the protein angle sequences. The differences between PAST and our PSIST methods are that we use different feature representation, we search for maximal matches and chain those matches, and finally our experimental setup is entirely different.

Most suffix tree algorithms are not designed to scale as efficiently when the input sequence is extremely large. Also, the use of suffix links, which are a key feature in obtaining the linear construction time, can result in poor locality of reference [20, 28, 65]. To address these issues, several disk-based suffix tree algorithms have been proposed in the last few years. Some of the approaches [28, 31, 57, 65] completely abandon the use of suffix links and sacrifice the theoretically superior linear construction time in exchange for a better locality of reference.

Trellis [47] proposes to develop a novel disk-based suffix tree algorithm without entirely sacrificing the suffix links structure. Trellis uses a novel approach to external suffix tree construction using the idea of *partitioning and merging*. An illustration of trellis is shown in Fig. 2.4. First, the long input sequence is broken into smaller substrings, such that the suffix tree from each partition can fit entirely in memory. For each partition, the suffix tree is constructed using Ukkonen's linear time algorithm [67], which also retains the suffix links. As each partition is processed its suffix tree is written to disk. Next trellis merges the suffix trees from each partition that share a common prefix. Trellis proposes

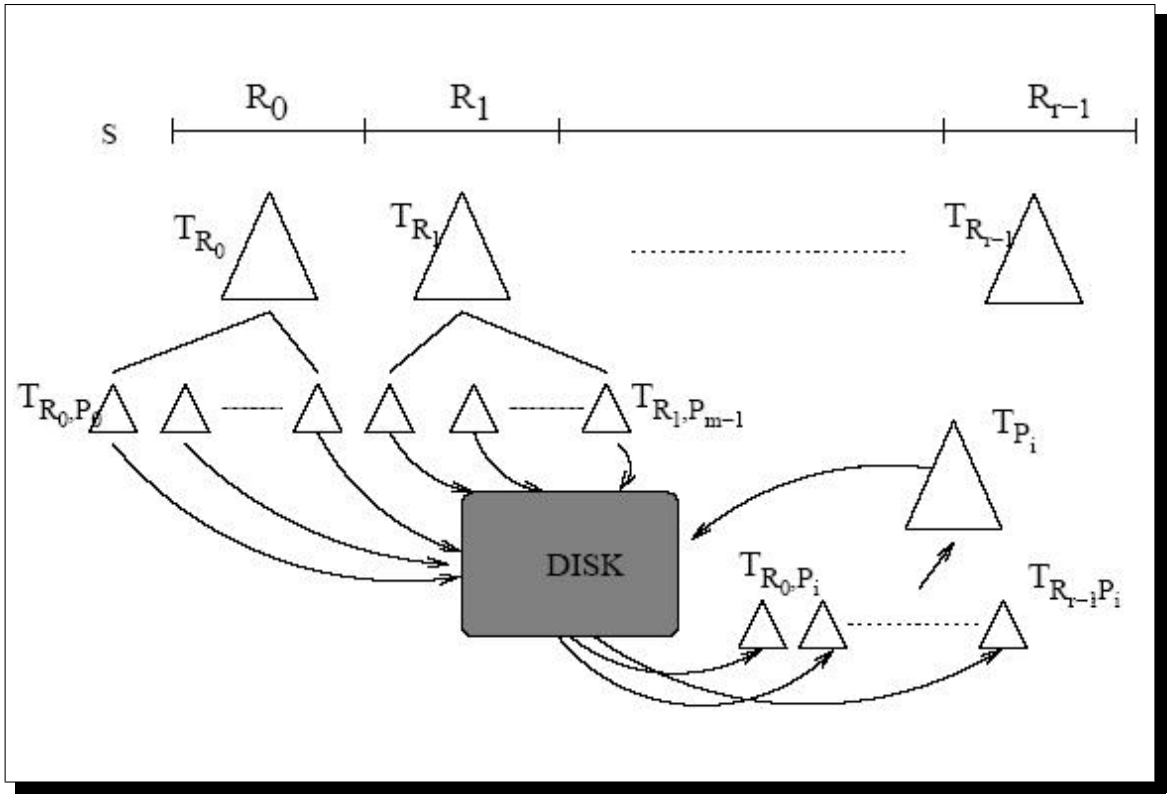

Figure 2.4: External Suffix Tree Construction

the use of "variable-length prefixes" such that each of the merged suffix sub-trees (of suffixes that share the common prefix) fits entirely in memory. The use of variable length prefixes also avoids the data skew problem (referring to the fact that not all prefixes are equally likely to appear in protein sequences) and eliminates the need to use any explicit buffer management policy to manage the tree nodes during construction.

A linear algorithm to construct distributed suffix trees (DST) was proposed in [12]. DST introduces a new notion of sparse suffix links and uses different rules to follow a sparse suffix link to the tree root. The suffix tree can be distributed over a number of computing nodes. It can handle larger data than existing suffix trees, but it does assume that the input data cannot exceed the size of real memory.

CHAPTER 3

Protein tertiary structure indexing using suffix trees

In this chapter, we present a new local feature representation of protein structures and convert the structure indexing problem into a sequence indexing task. We also propose a novel use of suffix trees to find the maximal matches between structure-feature sequences (SF-sequences) and use the alignment between the query and database SF-sequences to measure the structure similarity.

3.1 Overview

There are two steps in our method, indexing and querying. In the indexing step, we first extract the local features using a sliding window along the backbone. Each feature vector is a combination of the distance and angle, which have different measures. After normalization, the feature vector is converted to a multi-dimensional vector, which can be indexed by an R*-tree. To improve the performance, the feature vector is further discretized. Thus, the protein is converted to a sequence of symbols. We use suffix trees to index the protein sequence.

In the querying step, all the maximal matches are retrieved from the suffix tree first, then the alignment scores between the query and each template protein in the database are obtained by chaining the maximal matches using a fast greedy algorithm. Finally, we select the proteins based on the alignment scores which are obtained by running an accurate alignment algorithm: dynamic programming.

3.2 Indexing proteins

3.2.1 Local feature extraction

For a given query, we need to retrieve all the proteins that are similar locally, so each protein should be represented by its local features. In this section, we will extract the local features from the protein tertiary structure.

A protein is composed of an ordered sequence of residues linked by peptide bonds. Each residue has C_α, N and C atoms, which constitute the backbone of the protein. Although the backbone is linear topologically, it is very complex geometrically. Bond lengths, bond angles and torsion angles completely define the conformation and geometry of the protein.

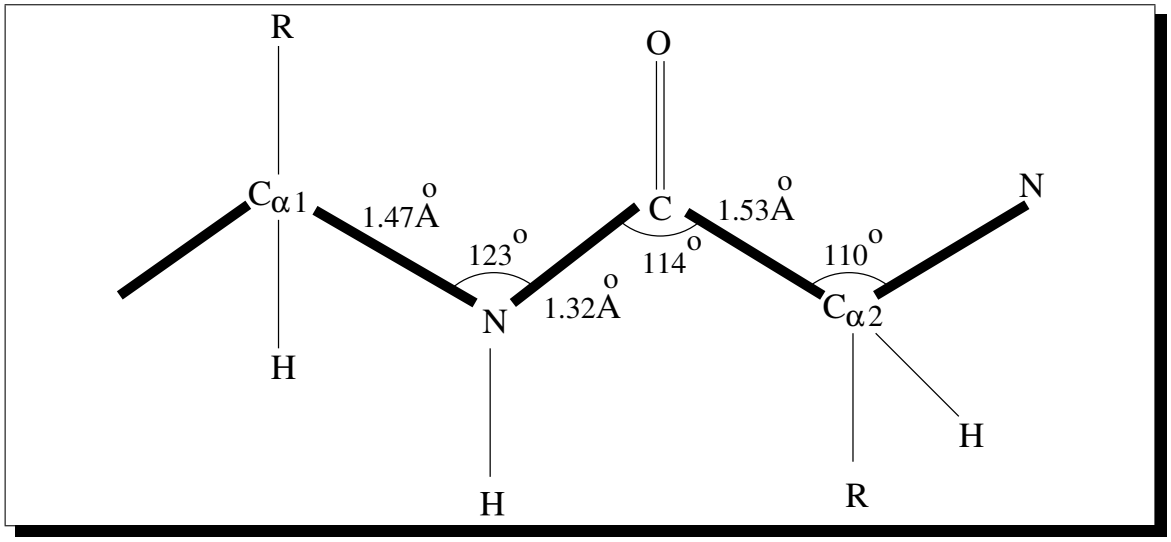

Figure 3.1: Bond length and bond angles

Bond length is the distance between the bonded atoms, and the bond angle is the angle between any two covalent bonds that include a common atom (see Figure 3.1). For instance, the bond length of N-C is 1.32Å ($\mathring{A}$ denotes distance in angstroms), the bond angle between C_α-N and N-C is 123°. Torsion angles are used to describe conformations around rotatable bonds (see Figure 3.2). Assume four consecutive atoms are connected by three bonds b_{i-1}, b_i and b_{i+1}. The torsion angle of b_i is defined as the smallest angle

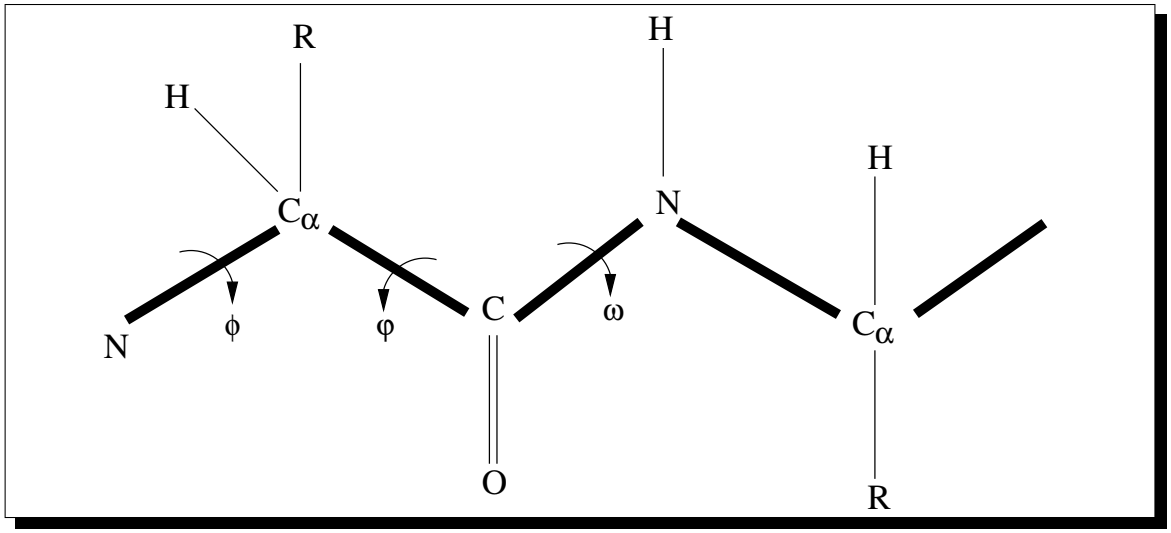

Figure 3.2: Torsion angles

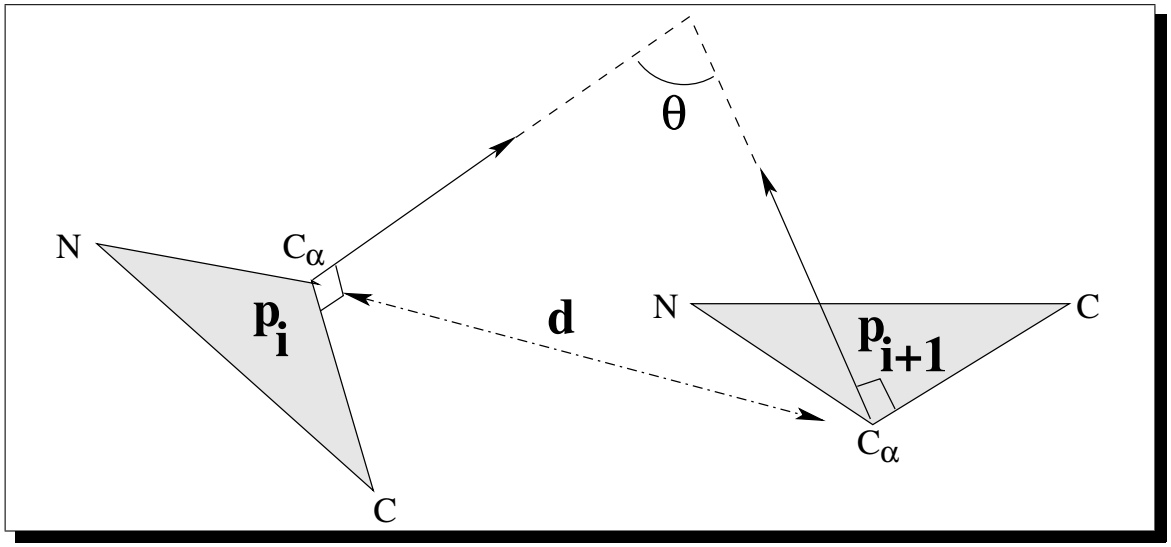

Figure 3.3: The distance and angle between two residues

between the projections of b_{i-1} and b_{i+1} on the plane perpendicular to bond b_i. In Figure 3.2, ϕ, ψ and ω are the torsion angles on the bonds N-C_α,C_α-C and C-N respectively.

To capture the local features more accurately, we need to extract the features from a set of local residues. To obtain the local feature vector, we first represent each residue individually, and then consider the relationship between a pair of residues and a set of residues. For each residue, the length of the C_α-N bond is $1.47\mathring{A}$ and that of the C_α-C

bond is $1.53\mathring{A}$, and the angle between the C_α-N and C_α-C bonds is 110°. Thus all the triangles formed by N-C_α-C atoms in each residue are equivalent, and each residue can be represented by a triangle of the same size.

The relationship between a pair of residues in 3D (three-dimensional) space can be fully described by the rigid transformation between two residues, which is a vector of 6 dimensions, containing 3 translational and 3 rotational degrees of freedoms. To reduce the dimension of the vector, we use a distance and an angle to describe the transformation features between two residues.

We define the distance d between a pair of residues as the Euclidean distance between their C_α atoms. The angle θ between a pair of residues is defined as the angle between the planes that contain N-C_α-C triangles representing each residue (see Figure 3.3).

The distance and angle are invariant to displacement and rotation of the protein. The Euclidean distance between two C_α atoms is calculated by their 3D coordinates directly. The angle between the two planes defined by the N-C_α-C triangles is calculated between their normals having C_α as the origin. The normal of the plane define by the triangle N-C_α-C is given as

$$\overrightarrow{n} = \frac{\overrightarrow{NC_\alpha} \times \overrightarrow{C_\alpha C}}{\|\overrightarrow{NC_\alpha} \times \overrightarrow{C_\alpha C}\|}$$

The angle between the two normals $\overrightarrow{n1}$ and $\overrightarrow{n2}$ is then calculated as

$$\cos\theta = \frac{\|\overrightarrow{n1}\|^2 + \|\overrightarrow{n2}\|^2 - \|\overrightarrow{n2} - \overrightarrow{n1}\|^2}{2\|\overrightarrow{n1}\|\|\overrightarrow{n2}\|}$$

To describe the local features between a set of residues, we slide a window of length w along the backbone of the protein. The distances and angles between the first residue i and all the other residues j (with $j \in [i+1, i+w-1]$) within the window are computed and added to a feature vector. Each window is associated with one feature vector.

Let $P = \{p_1, p_2, \ldots, p_n\}$ represent a protein, where p_i is the ith-residue along the backbone. The feature vector of the protein is defined as $P^v = \{p_1^v, p_2^v, \ldots, p_{n-w+1}^v\}$, where w is the sliding window size, and p_i^v is a feature vector

$$\{d(p_i, p_{i+1}), \cos\theta(p_i, p_{i+1}), \ldots, d(p_i, p_{i+w-1}), \cos\theta(p_i, p_{i+w-1})\}$$

where $d(p_i, p_j)$ is the distance between the residues p_i and p_j, and $\theta(p_i, p_j)$ gives the angle between the residues p_i and p_j. With window size w, the dimension of each feature vector p_i^v is $2(w-1)$.

3.2.2 Normalization

Our feature vector is a combination of distances and angles, which have different measures, so a normalization procedure is needed after the feature vectors are extracted. After normalization, each value of the feature vector is within $[0, 1]$, which is a point in the multiple dimensional space and can be indexed using an R^*-tree. To find the match segments between the query and template protein using a suffix tree, we need to discretize the real value of the feature vector to a number of bins, which is indexable for suffix trees.

For normalizing the distances, we need to know the upper-bound and lower-bound on the distance between the i-th and $(i+w-1)$-th residue in the protein. From figure 3.1, the average distance between $C_{\alpha 1}$-N atoms is $d_1 = 1.47\mathring{A}$, the average distance between N-C atoms is $d_2 = 1.32\mathring{A}$, and the angle α between $C_{\alpha 1}$-N and N-C bonds is 123°. The distance between $C_{\alpha 1}$-C atoms is therefore $d(C_{\alpha 1}, C) = \sqrt{d_1^2 + d_2^2 - 2d_1 d_2 \cos\alpha} = 2.453$. The distance between C-$C_{\alpha 2}$ atoms is $d(C, C_{\alpha 2}) = 1.53$, so the average distance between two C_α atoms is: $d(C_{\alpha 1}, C_{\alpha 2}) \leq d(C_{\alpha 1}, C) + d(C, C_{\alpha 2}) = 2.453 + 1.57 = 4.023$. If the distance between two atoms is greater than 4.023, it is trimmed to 4.023. For a sliding window of size w, the lower bound of the distance between any two atoms is 0, and the upper bound is $4.023s(w-1)$, so the distance between any pair of residues within

a w length window is in the range $[0, 4.023(w-1)]$. The angle θ is in the range $[0, \pi]$, and $\cos\theta \in [-1, 1]$.

All the distances and angles are normalized and binned into an integer within the range $[0, b-1]$ for some integer b. We use the equation $d = \lfloor \frac{db}{4.023(w-1)} \rfloor$ to normalize and bin the distance and $\cos\theta = \lfloor \frac{(\cos\theta+1)b}{2} \rfloor$ to normalize and bin the angle. Table 3.1 shows 3 examples of normalized and binned feature vectors for $w = 3$ and $b = 10$. The size of each feature vector is $2(w-1) = 4$, and the normalized value is within $[0, 9]$.

Table 3.1: Examples of normalized feature vectors for $w = 3$ and $b = 10$

	Feature vector			
	d	$\cos\theta$	d	$\cos\theta$
original	3.55	0.29	5.4	-0.23
normalized	4	6	6	3
original	4.04	0.11	5.75	-0.25
normalized	5	5	7	3
original	3.60	0.45	5.29	0.21
normalized	4	7	6	6

After normalization and binning, each feature vector is defined as $p^s = \{p_0^s, p_1^s, \ldots, p_{2(w-1)-1}^s\}$, where p_i^s is an integer within the range $[0, b-1]$. Thus, the structure of each protein P is converted to a structure-feature sequence $P^s = \{P_0^s, P_1^s, \ldots, P_{n-w+1}^s\}$, called the *SF-sequence*, where P_i^s is the i-th normalized feature vector (p^s) along the backbone. Note that each symbol within an SF-sequence is a vector of length $2(w-1)$, to which we assign a unique integer identifier as its label. Thus the SF-sequences are over an alphabet of size $b^{2(w-1)}$.

3.2.3 Generalized suffix trees construction

After obtaining the SF-sequences for all proteins in the database, we use a Generalized Suffix Tree (GST) as the indexing structure. GST is a compact representation of the

suffixes of sequences, and can be constructed in linear time [67]. A suffix can be located by following an unique path from the root to a leaf.

To save the storage space of the suffix tree, we map each structure feature vector p^s to a unique key or symbol for the suffix tree construction, and map it back to the normalized vector when we compute the distance between two feature vectors. For instance, the three feature vectors in table 3.1 could be mapped to the symbols a, b and x respectively.

Notation: Letting GST be a generalized suffix tree, we use the following notation in the rest of the section. We use N for a node in the suffix tree, E for an edge, $C(E)$ for a child node of the edge E, $L(E)$ for the label on edge E, $L(E[i])$ for the i^{th} symbol of edge label $L(E)$, $P(N)$ for the path-label of node N (formed by concatenating all the edge labels from the root node to N), and $P(E[i])$ for the path-label of $L(E[i])$. Further, each leaf node in GST contains a sequence-position pair (x, p), where x is a sequence identifier, and p is the start position of the suffix within sequence x. For any node N, we use the notation *sp-list*(N) for the collection of the sequence-position pairs for all the leaves under N.

Example: Figure 3.4 shows an example of GST for two SF-sequences $S_1 = xabxa$ and $S_2 = babxba$, over the alphabet $\{a, b, x\}$, obtained by mapping each normalized feature vector in table 3.1 to a unique letter symbol. Node 0 is the root node, node 1 to 7 are internal nodes, and the rest are leaves. '$' is the unique termination character. If '$' is used as a edge label, it only indicates the end of the suffix, so it is not part of the path label. For example, the path label of node 12 is a. The purpose of using '$' as an edge label is to distinguish internal nodes from leaf nodes, and only leaf nodes contain the sequence-position pairs.

In figure 3.4, the path label of node 7 is xa. The edge label $L(E)$ of the edge out of node 7 is bxa, so its second character $L(E[2])$ is x, and its path-label $P(E[2])$ is $xabx$.

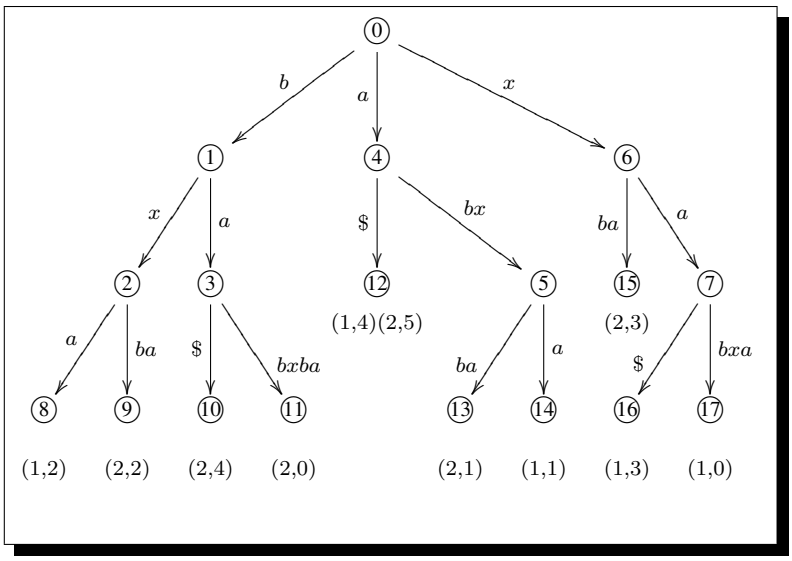

Figure 3.4: GST for sequences $S_1 = xabxa$ and $S_2 = babxba$

The sequence-position identifier $(1, 0)$ of node 7 stands for $xabxa$, a suffix of sequence S_1 that starts at position 0. Thus *sp-list*$(7) = \{(1, 0), (1, 3)\}$, and the sp-list for node 6 is *sp-list*$(6) = \{(2, 3), (1, 3), (1, 0)\}$.

3.2.4 Complexity

After converting the protein to a sequence of discretized feature vectors, we can use the algorithm presented in [67] to build the generalized suffix tree for the database. Thus, the time and space complexities of the suffix tree construction are $O(N|\Sigma|)$, where N is the total size of the database, and Σ is the size of the alphabet.

3.3 Querying

So far we have discussed how to build the suffix tree indexing based on the local structure features for each protein. In this section, we will present how to search for similar proteins.

Given a query (Q, ϵ), we first extract its feature vectors and convert it into a SF-sequence Q^s as described in section 3.2.1 and 3.2.2. Then three phases are performed:

searching, ranking and post-processing. The searching phase retrieves all the matching segments/subsequences from the database within a distance threshold ϵ (on a per symbol basis), the ranking phase ranks all the proteins by chaining the matching segments. Since the ranking step can only find the alignment between the query and the template proteins approximately based on maximal matches, the ranking of the similar proteins is not optimal. Thus, we also perform a post-precessing step, which uses the Smith-Waterman [61] approach to find the best local alignment between the query and the selected proteins.

3.3.1 Searching

For a given query SF-sequence $Q^s = \{Q_1^s Q_2^s \dots Q_n^s\}$, maximum feature distance threshold ϵ, and minimum match length threshold l, the search algorithm finds all maximal matching SF-subsequences $P^s = \{P_1^s, P_2^s \dots P_m^s\}$ that occur in both the query SF-sequence and any database protein SF-sequence. A maximal match has the following properties:

1. There exists a matching SF-subsequence $Q_{i+1}^s \dots Q_{i+m}^s$ of Q^s, such that $dist(Q_{i+j}^s, P_j^s) < \epsilon$, where $j = 1, 2, \dots, m$, and Q_{i+j}^s and P_j^s are the normalized and binned feature vectors of length $2(w-1)$. The distance function used in our algorithm is Euclidean distance.

2. The length of the match is at least as long as the length threshold, i.e., $m \geq l$.

3. Assume P^s is a SF-subsequence of protein R^s, then neither $P^s v$ nor $v P^s$ is a matching SF-subsequence of Q^s and R^s for any feature vector v (this ensures maximality).

For instance, assume $\epsilon = 0$ and minimum length threshold $l = 3$, then abx is a maximal match between the SF-sequences $xabxa$ and $babxba$ of Figure 3.4 because the characters before and after the abx are (x,b) and (a,b) respectively, and neither of them match.

Note that our approach differs from the MUMmer genome alignment method presented in [13] which finds *exact* maximal *unique* matches between *two* genomes.

Input : query Node N_q, database Node N_d, distance ϵ, length threshold l
Output : maximal matches set ($MMSet$)
Initialization: $MMSet = \emptyset$

Procedure: MMS(N_q,N_d,ϵ,l)
foreach *edge* E_q *out of* N_q **do**
  foreach *edge* E_d *out of* N_d **do**
    NS($E_q, 0, E_d, 0, \epsilon, l$).

Figure 3.5: MaximalMatchesSearch algorithm

To find all maximal matches within ϵ between the query Q^s and suffix tree GST_d built from the database proteins, one solution is to trace every SF-subsequence of Q^s from the root of GST_d, but the common prefix of two subsequences will be searched twice and more comparisons will be performed. To reduce the number of comparisons, we build another suffix tree GST_q for Q^s, and then traverse two suffix trees simultaneously to retrieve all the maximal matches. In the discussion below, we use the subscript q for the query and d for the database. For instance, N_q stands for a query suffix tree node, while N_d stands for a database suffix tree node.

The matching algorithm starts with the MMS procedure as shown in Figure 3.5, and its inputs are the root node (N_q) of the query suffix tree GST_q, the root node (N_d) of the database suffix tree GST_d, the distance tolerance ϵ and the minimum length of the maximal match l. For every edge out of the query node and database node, MMS calls the NodeSearch procedure (see Figure 3.6) to match their labels and follow the path to find all the matching nodes.

In the NodeSearch procedure, for two edges from different suffix trees, the distance between the corresponding pair of label symbols $L(E[i]_q)$ and $L(E[j]_d)$ is computed in step 2. If the distance is larger than ϵ, which implies a mismatch, the procedure updates the $MMSet$ and proceeds to the next branch. If there is no mismatch, the short edge

will reach the end first. If the child node of the short edge is a leaf, we need to update the $MMSet$. If the child node is an internal node, two different procedures are called recursively. 1) If the lengths of two edge labels are the same, then the MMS procedure is called for two child nodes in step 3. 2) If one of the edges has a shorter label, the algorithm NodeSearch will be called recursively with the new input of all the edges out of the child node of the short edge (please see steps 4 and 5).

Each matching SF-subsequence s is defined by two triplets (x, p, l) and (y, q, l), where p and q are the start positions of s in the query sequence Q_x and the protein sequence P_y respectively, and l is the length. If s is a maximal match, it will be added to the $MMSet$ in the $updateMMS$ procedure. To identify a maximal match, we need to compare whether any extension of the match will result in a mismatch. In our algorithm, each common subsequence s is obtained either from characters mismatch or a leaf node, so we just need to compare the characters before the common subsequence ($Q_x[p-1]$ and $P_y[q-1]$) to identify the maximal match.

We can also process multiple query SF-sequences at the same time by inserting them to the query suffix tree GST_q, so the nodes with the same path-label are visited only once and performance will be improved.

Example: Figure 3.8 is the GST for a query sequence $q1 = abab$. We'll show how to traverse the query suffix tree (figure 3.8) and database suffix tree (figure 3.4) together to find all the maximal matches. In our example, we assume that the distance threshold $\epsilon = 0$ (which implies exact matching) and maximal match length threshold $l = 3$.

We start with two root nodes 0 and $\bar{0}$ (MaximalMatchesSearch algorithm). Node $\bar{0}$ of the query suffix tree has two out edges, and node 0 of the database suffix tree has three out edges. For all the edge pairs, we compare the distance between their labels. We consider the leftmost edges of the trees ($\bar{0}$-$\bar{1}$ and 0-1) first, and find that their labels are b and match with each other. Both edges have the same length and reach their end at the

Input : query Edge E_q, query Edge iterator i, database Edge E_d, database Edge iterator j, distance ϵ, length threshold l
Output : maximal matches set ($MMSet$)

Procedure: NS(E_q,i,E_d,j,ϵ,l)
1 **while** $i < L(E_q).len$ **and** $j < L(E_d).len$) **do**
2 **if** $dist(L(E[i]_q), L(E[j]_d)) > \epsilon$ **then**
updateMMS($C(E_q), C(E_d), P(E[i]_q).len - 1, l$).
return;
else
$i = i + 1, j = j + 1$
3 **if** i = $L(E_q).len$ **and** j = $L(E_d).len$ **then**
if *isleaf*($C(E_q)$) **or** *isleaf*($C(E_d)$) **then**
updateMMS($C(E_q), C(E_d), P(E[i]_q).len - 1, l$)).
else
MMS($C(E_q), C(E_d), \epsilon, l$).
4 **if** i = $L(E_q).len$ **and** $j < L(E_d).len$ **then**
if *isleaf*($C(E_q)$) **then**
updateMMS($C(E_q), C(E_d), P(E[i]_q).len - 1, l$).
else
foreach *edge* E_C *out of* $C(E_q)$ **do**
NS(E_C,0,E_d,j,ϵ,l).
5 **if** $i < L(E_q).len$ **and** j = $L(E_d).len$ **then**
if *isleaf*($C(E_d)$) **then**
updateMMS($C(E_q), C(E_d), P(E[j]_d).len - 1, l$).
else
foreach *edge* E_C *out of* $C(E_d)$ **do**
NS($E_q, i, E_C, 0, \epsilon, l$).

Figure 3.6: NodeSearch algorithm

same time (step 3 of NodeSearch algorithm). Since their child nodes ($\bar{1}$ and 1) are internal nodes, we continue to compare the edges out of the child nodes. The leftmost edge out of the child node $\bar{1}$ of the query suffix tree is $\bar{1}$-$\bar{2}$ and its label is ab. The leftmost edge out of the child node 1 of the database suffix tree is 1-2 and its label is x. The first symbols a and x do not match (step 2 of NodeSearch algorithm), the matching subsequence so far

```
Input  : query Node N_q, database Node N_d, match
         length m, length threshold l
Output : maximal matches set (MMSet)

Procedure: UpdateMMS(N_q, N_d, m, l)
if m >= l then
    foreach (x, a) ∈ sp-list(N_q) do
        foreach (y, b) ∈ sp-list(N_d) do
            if dist(Q_x[a − 1], P_y[b − 1]) > ε then
                add ((x, a, a + m − 1), (y, b, b + m − 1)) to
                MMSet
```

Figure 3.7: UpdateMaximalMatchesSet algorithm

is "b" and its length is less than the threshold 3, so we proceed to the next branch (1-3) of the database suffix tree. The second edge 1-3 of the child node 1 has the label a, which matches the first symbol of the query edge $\bar{1}$-$\bar{2}$. Now the edge 1-3 reaches its end (step 5 of NodeSearch algorithm); we continue to compare the next edge 3-11 with $\bar{1}$-$\bar{2}$ and find that the first symbol of edge 3-11 and the second symbol of edge $\bar{1}$-$\bar{2}$ is b. The edge $\bar{1}$-$\bar{2}$ reaches a leaf node (step 4 of NodeSearch algorithm), and the matching subsequence is "bab" with length is equal to the threshold l. After checking (UpdateMaximalMatchSet algorithm), we find that "bab" is a maximal match, so its representation $\{$(2,0,3), ($\bar{1}$, 1, 3)$\}$ is added to the maximal match set. We keep on processing the remaining edges until all the branches are scanned and all the maximal matches are found.

3.3.2 Ranking

The maximal matches are obtained for the query sequence and reference sequences in the database. Every maximal match is a diagonal run in the matrix formed by a query and reference sequence. We use the best diagonal runs described in the FASTA algorithm [45] as our ranking scheme. We calculate the alignment as a combination of the maximal matches with the maximal score. The score of the alignment is the sum of

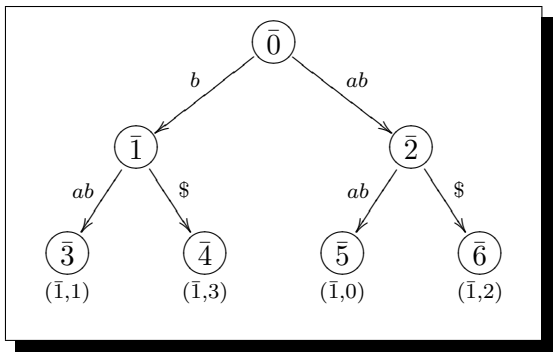

Figure 3.8: GST for sequences $Q_{\bar{1}} = abab$

the scores of the maximal matches minus the gaps penalty. Both the score of a maximal match and a gap are their lengths in our algorithm. Two maximal matches can be chained together if there are no overlap between them. We use a fast greedy algorithm to find the chains of maximal alignments. At first, the maximal matches are sorted by their length. The longest maximal match is chosen first, and we remove all other overlapping matches. Then we choose the second longest maximal match which doesn't overlap with the longest match, remove its overlapping matches and repeat the above steps until no maximal matches are left. This way we find the longest chained maximal matches between the query and each retrieved database SF-sequence. The alignment score between the query and a template structure-sequence is the highest score of the chained maximal matches, which is the score of the longest chained maximal matches in our experiments. After obtaining the alignment scores for each protein, the proteins are sorted by their alignment scores. Finally all the candidates with small alignment scores are screened out and only the top similar proteins are selected.

3.3.3 Post-processing

Each top protein SF-sequence with a high score selected from the database is aligned with the query by running the Smith-Waterman [61] dynamic programming method. The similarity score between two residues is set to 1 if the distance between their normalized

feature vectors is smaller than ϵ, or it is 0. Proteins are then ranked in decreasing order according to their new alignment scores and the top proteins with the highest scores are reported to the user.

3.3.4 Complexity

The number of maximal matches found in the searching step is far smaller than the size of the protein so the ranking step is much faster compared with the searching step and we only analyze the running time of searching for maximal matches.

The complexity of searching depends on the distance ϵ which decides whether two symbols match or not. We consider two extreme cases here.

- Assume $\epsilon = 0$, the symbols match if and only if they are identical. Thus, searching the whole query sequence is equivalent to the exact sequence matching [67], with complexity $O(m|\Sigma|)$, where m is the length of the query. In our algorithm, if we do a suffix tree search once for each suffix of the query, the complexity will be $O(n|\Sigma|) + O((n-1)|\Sigma|) + \ldots + |\Sigma| = O(n^2|\Sigma|)$. To reduce the complexity, we build a suffix tree for the query. The number of edges in the query suffix tree is at most $2m|\Sigma|$, and each edge is searched only once, so the complexity of the query sequence searching will be $O(2m|\Sigma|) = O(m|\Sigma|)$.

  In our algorithm, we also need to traverse the subtree at the end of the matching path using any linear-time traversal to collect all the maximal matches. Assume k is the total number of occurrences of all maximal matches in the database suffix tree and l is the total number of occurrences of all maximal matches in the query suffix tree, the complexity will then be $O(m|\Sigma| + k + l)$.

- If ϵ is larger than the maximum distance between any two symbols, any edge labels will result in a match. Thus, for each edge of the query suffix tree, it will match all the edges in the database suffix tree, and the complexity will depend on the size of

database suffix tree. The worst case is that for each path of the query suffix tree, the whole database suffix tree is scanned once, so the complexity is $O(mN)$, where m is the query length and N is the total length of the database.

3.4 Experiments

To evaluate the performance of our algorithm we conduct two sets of experiments. The first test evaluates the performance of in-memory suffix tree, while the second tests the external suffix tree.

3.4.1 In-memory suffix tree experiments

The SCOP database [39] classifies proteins according to a four level hierarchical classification, namely, family, super-family, fold and class. Proteins are clustered together into families on the basis of either significant sequence similarity (35%) or high structural similarity and some sequence identity. Families are placed together in superfamilies if their proteins have low sequence identities but high structure similarity; superfamilies and families are defined as having a common fold if their proteins have the same major secondary structures in the same arrangement with the same topological connections. Most of the folds are assigned to one of the five structural classes on the basis of the secondary structures of which they are composed: all alpha, all beta, alpha and beta, alpha plus beta and multi-domain. As of October 2004, (the last time it was updated) the SCOP database contains a total of 25973 PDB entries, 2845 families, 1539 super-families, and 945 folds. The SCOP database has been created by manual inspection and aims to provide a detailed and comprehensive description of the structural and evolutionary relationships between all proteins whose structure is known. It provides a broad survey of all known protein folds. Since the SCOP database is curated by visual inspection it is considered to be extremely accurate. For our tests, the target database we used has proteins from four classes of SCOP: all α, all β, $\alpha + \beta$ and α/β. Our dataset D includes a total of

1810 proteins taken from 181 superfamilies which have at least 10 proteins, but only 10 proteins are chosen from each superfamily. One protein from each superfamily is chosen randomly as the query, so the size of the query set D_q is also 181. This is the same dataset used in several previous indexing studies [5, 9].

To evaluate our algorithm we perform two different tests: The *retrieval* test finds the number of correct matching structures from the same superfamily as the query among the top k scoring proteins, and the *classification test* tries to classify the query at the superfamily and class levels. Our algorithm is implemented in C++ and all experiments reported below were done on a PC with a 2.8GHz CPU and 6GB RAM, running Linux 2.6.6 and the GNU g++ compiler 3.3.2 with optimization level -O.

3.4.1.1 Retrieval test

We compare our approach with one of the best previous indexing approaches ProGreSS [5], using the Java-based code provided by its authors. Since our code is implemented in C++, it's difficult to say whether runtime measurements are as comparable as they would be if both implementations were in the same language. We also directly compare with a geometric hashing based [33] indexing method, which we coded ourselves. For geometric hashing we take two consecutive C_α atoms along the backbone as the reference frame. Each remaining C_α atom and the reference frame form a triplet. The three pair-wise distances from a triplet are added to an R^*-tree if all of them are within $7\mathring{A}$. For querying, we form query triplets in the same manner, and find all matching triplets within ϵ range. Suppose there are n triplets with the same query reference frame, and the matching protein has m triplets with the same reference frame, then these two reference frames are considered to be a matching pair if the ratio between m and n is greater than a threshold, i.e., if $m/n > 0.75$. The score of a protein is its number of matching reference frames with respect to the query, and the proteins are ranked based on their scores.

We ran the experiments using PSIST, ProGreSS, and geometric hashing, to obtain

the number of proteins found from the same superfamily for each of the 181 queries. Since each superfamily has 10 proteins, including the query, there can be at most 10 correct matching proteins from the same superfamily.

There are five parameters used in our approach. w is the size of the window used to index the local features, b is the range used to normalize the feature vectors, ϵ is the distance threshold based on the normalized feature vectors, l is the minimum length of the maximal matches, and k is the number of top scoring proteins reported. We first show how PSIST performs for different values of w, ϵ, b, l and k.

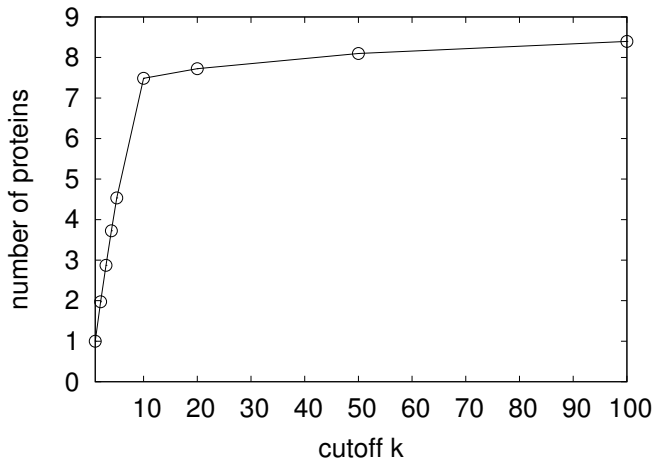

Figure 3.9: Number of proteins found from the same superfamily for different top-k value ($w = 3$, $b = 10$, $\epsilon = 3$ and $l = 10$).

Figure 3.9 shows the number of proteins found from the same superfamily for different top-k cutoffs. Note that the number of correct matches is an average over all 181 SCOP superfamilies used in our test. The retrieval performance tapers off as k increases. We choose the largest cutoff as $k = 100$, since there is not much to be gained by using larger values.

We next study the effect of varying window size w, while keeping $b = 10$, $\epsilon = 3$ and $l = 15$. Figure 3.10 shows that a smaller window size of $w = 3$ yields the highest number of correct matches (on average 8 correct matches out of 10), and the retrieval rate

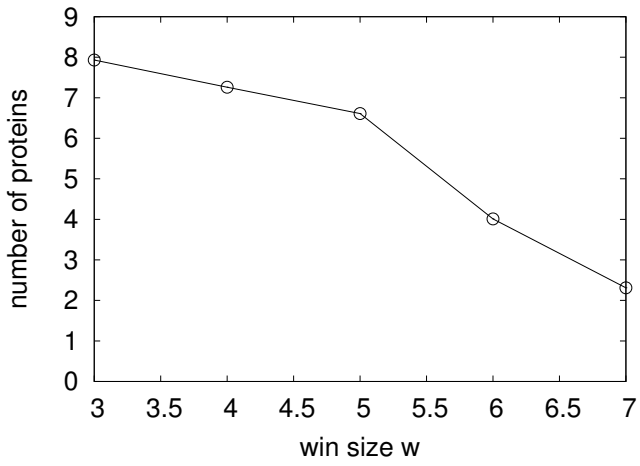

Figure 3.10: Number of proteins found from the same superfamily for different window sizes w when ($b = 10$, $\epsilon = 3$ and $l = 15$)

drops as w increases. For a smaller window size more matches are found in the database within the ϵ distance, and PSIST is able to find the best matches after finding the chain of maximal matches. For larger windows the number of matches drops and some of the correct proteins are missed. From this experiment we conclude that $w = 3$ is the best for PSIST.

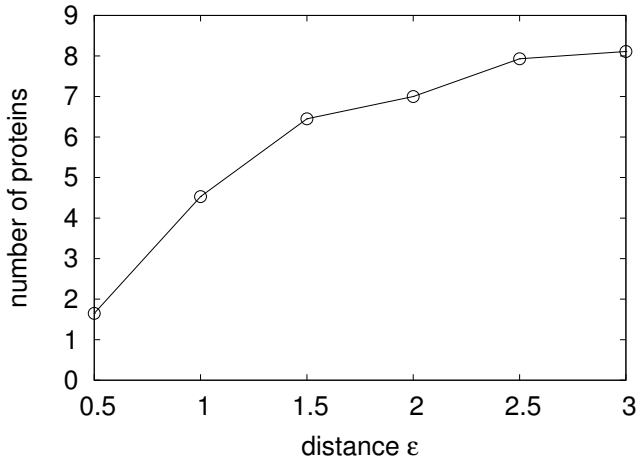

Figure 3.11: Number of proteins found from the same superfamily for different ϵ ($w = 3$, $b = 10$ and $l = 15$)

Figure 3.11 shows the effect of varying ϵ with $k = 100$. The larger the ϵ, the more

the structures retrieved and then PSIST is able to find the correct ones by ranking the alignments. We find that $\epsilon = 3$ works well for PSIST, and performance tapers off for larger values.

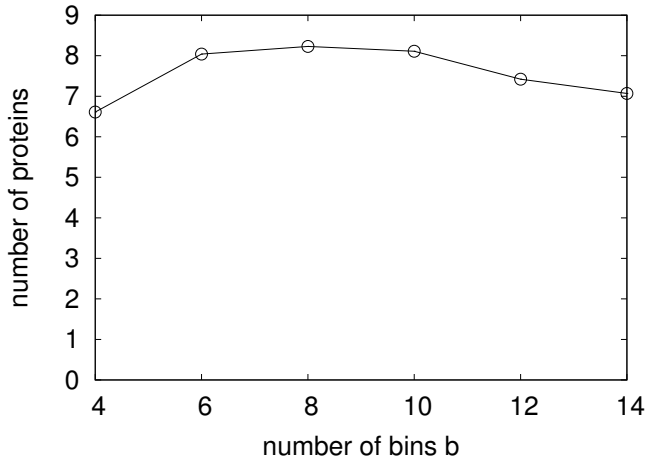

Figure 3.12: Number of proteins found from the same superfamily for different b ($w = 3$, $\epsilon = 2.5$ and $l = 15$)

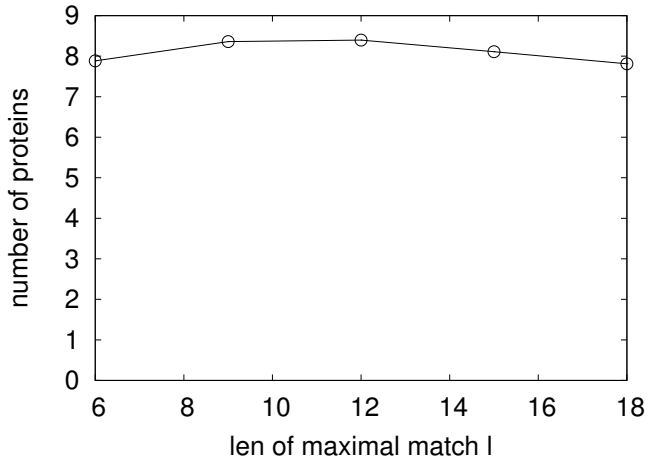

Figure 3.13: Number of proteins found from the same superfamily for different length of maximal matches ($w = 3$, $\epsilon = 2.5$ and $b = 10$)

Figure 3.12 and 3.13 show that the varying normalization range b and the length of maximal match l have a similar effect on the number of proteins found from the same superfamily. For smaller range b and maximal match length l, there can potentially be

many incorrect proteins with similar match segments, but for larger b and l, fewer maximal matches, but correct proteins are found. PSIST obtains its best performance when the bin range is between 6 and 10, and the length between 9 and 12.

Table 3.2: Overall comparison of the number of proteins found from the same superfamily among the top k candidates

Algorithm	top4	top10	top50	top100
GeoHash	2.43	3.74	4.40	4.86
ProGreSS	3.53	6.17	6.69	7.09
PSIST	3.72	7.49	8.10	8.40

Table 3.2 shows the comparison of the number of proteins found from the same superfamily for different top k values. The table compares the performance of our approaches against geometric hashing and ProGreSS. Geometric hashing can find only 2.43 correct proteins within the top 10 proteins (with $\epsilon = 0.18$, which was the best value we determined empirically). It also has relatively poor performance for other values of k. Both ProGreSS and PSIST retrieve more than 3 correct proteins within the top 4 candidates. However, PSIST performs better than ProGreSS when the cutoff increases. For instance, PSIST could find 7.49 out of 10 proteins within the top 10 candidates. Note that based on the previous experiments, for the PSIST algorithm we set $w = 3$, $b = 10$, $\epsilon = 3$ and $l = 9$. For fair comparison, we tuned the parameter settings for ProGreSS to report its best results (we use sequence distance threshold $\epsilon_t = 0.05$, the structure distance threshold $\epsilon_q = 0.01$ and window size $w = 3$).

3.4.1.2 Classification test

In the classification test, we assume we do not know the superfamily or the class to which a query protein belongs. For each query we then classify it into one of 181 SCOP superfamilies and one of the four SCOP classes (all α, all β, $\alpha + \beta$ and α/β) as

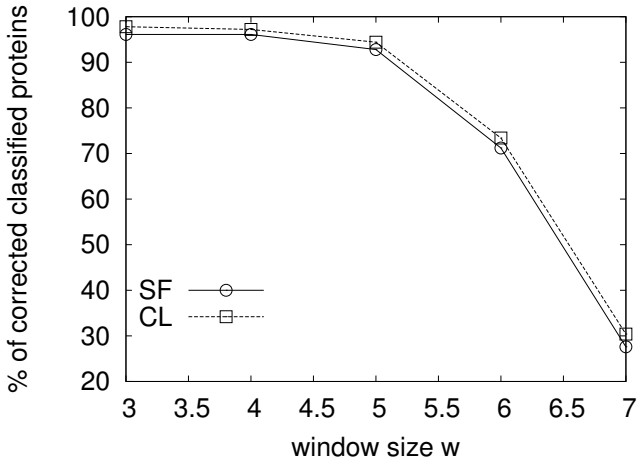

Figure 3.14: Percentage of query proteins correctly classified for different window sizes when ϵ=3

follows. For each query, the top k similar proteins are selected from the database. The query itself is not counted in the top k matches. Each protein among the top k matches is assigned a score, a superfamily id, and a class id. The scores of the top k proteins from the same superfamily or class are accumulated. The query is assigned to the superfamily or class with the highest score. This classification approach can thus be thought of as k Nearest Neighbor classification. Below we report results separately for the superfamily-level and class-level classification. For the performance, we report the percentage of correctly classified query proteins (out of the 181 queries). For the classification tests we also compare with the numbers reported by PSI [9] and LFF [11], in addition to the results of ProGreSS and Geometric Hashing. For PSIST, ProGreSS and Geometric Hashing we use the best parameter settings reported in the last section.

Proteins are classified correctly if the proteins from the same superfamily have a better rank. Thus the classification accuracy is proportional to the number of the correct proteins found in the top candidates. For instance, Figure 3.14 shows the percentage of query proteins correctly classified for different window sizes when $\epsilon = 3$, and using $k = 3$, at the superfamily (SF) and class (CL) levels. It has a similar shape as Figure 3.10; the more the proteins found from the same superfamily, the higher the accuracy obtained.

Table 3.3: SCOP classification accuracy comparison at the superfamily (SF) and class (CL) level

Algorithm	Superfamily	Class
Geometric Hashing	60.2%	72.9%
PSI	88%	N/A
LFF	68.6%	93.2%
ProGreSS	97.2%	98.3%
PSIST	97.8%	99.4%

Table 3.3 shows the SCOP classification comparison with other algorithms at the superfamily and class level respectively. Geometric hashing has the worst performance, it can only classify 60.2% and 72.9% proteins correctly at the superfamily and class level. PSI [9] uses SSE-based features, and its accuracy for superfamily is 88%, but its class accuracy is unavailable. LFF profiles [11] only classify 68.5% of the superfamily correctly, but it agrees with SCOP classification at 93% for class level (Note that LFF profiles use a different testing protein dataset than ours). ProGreSS and PSIST could obtain more than 3 proteins within the top 4 candidates, so their accuracy is very close and much better than the others. ProGreSS uses both the structure and sequence features to classify the proteins, and its accuracy is 97.2% and 98.3% at the superfamily and class level. Without considering the sequence features, PSIST has slightly better performance than ProGreSS: its accuracy is 97.8% and 99.4% at the superfamily and class level. We also classified the protein at the fold level using a similar method, and the accuracy of PSIST at the fold level was 97.2%.

3.4.1.3 Time performance test

We compare the running time of different approaches in this section. Suppose a protein has n residues, the window size is w, then the number of feature vectors is $n-w+1$, so the complexity of our approach is $O(n-w+1) = O(n)$ per protein. Assume the

average number of neighbors of each reference frame is k, the complexity of our implementation of geometric hashing is $O(kn)$. Although they have the same complexity, geometric hashing is slower because of the coefficient k; its running time is 1080.4 seconds per query for distance $\epsilon = 0.18$. Since this time does not compare favorably with either PSIST or ProGreSS, we do not report detailed timing comparisons for geometric hashing.

Table 3.4: Running time comparison

Algorithm	**SF%**	**CL%**	**top10**	**time(s)**
ProGreSS	97.2%	98.3%	6.17	1.67
PSIST-1	96.7%	98.3%	6.57	0.47
PSIST-2	97.2%	99.4%	7.19	4.41
PSIST-3	97.2%	99.4%	7.19	3.28

Both ProGreSS and PSIST provide a trade-off between the running time and the accuracy performance by adjusting the parameters such as window size and distance. For a fair algorithmic comparison, we compare the time performance of ProGreSS and PSIST based on their retrieval and classification test. Table 3.4 shows the running time for ProGreSS and PSIST. For ProGreSS, we choose the best sequence and structure distance thresholds and set window size $w = 3$. We set $w = 3$, $b = 2$, $\epsilon = 0$ and $l = 15$ for the first case of PSIST, and it is 3.5 times faster than ProGreSS with similar retrieval and classification performance. The last two cases have the same parameters: $w = 3$, $b = 6$, $\epsilon = 2$, $l = 15$, but the difference is that the third case builds a query suffix tree for every 20 queries and processes them together. They have the same retrieval and classification performance but the third case is faster. Although both cases are slower than ProGreSS, they retrieve on average more proteins (7.49 vs. 6.47) out of the top 10 matches and obtain slightly higher accuracy.

3.4.2 External suffix tree experiments

In this section, we conduct an extensive set of experiments using an external suffix tree: Trellis [47]. The first test compares the performance of PSIST with ProGreSS [5], a state-of-the-art protein indexing method. The second test compares the results of suffix tree indexing using different pieces of information: sequence or structure. The third test shows the performance of indexing the whole set of proteins in SCOP [39], a database of proteins classified according to their structure. Please note that we use different datasets and queries for all three tests, but all of them use external suffix tree.

The external suffix tree experiments reported below were done on a Power Mac G5 with 2.7GHz CPU, and 4GB Memory, running Darwin Kernel Version 8.0.0.

3.4.2.1 Comparison with ProGreSS

The CATH [42, 44] database gives a hierarchical classification of protein domain structures based on sequence and structure similarity. It operates on domains because domains are likely to be the fundamental evolutionary building blocks. CATH has four major levels of classification, namely Class, Architecture, Topology and Homologous family. Homologous family is the lowest level; it contains either proteins having significant sequence similarity (35%) or high structural similarity and some sequence identity (20%). The sequences are aligned using dynamic programming and the structures are aligned using SSAP [43]. Protein domains that share a significant structure similarity but low sequence similarity are grouped into the same Topology. Architecture is assigned manually according to the gross arrangement of secondary structures in 3D space. At the top of the hierarchy, domains are clustered into four classes automatically by the percentage of α-helices or β-strands. The latest version (2.6.0) of the CATH database contains more than 67,000 domains classified into 6,003 homologous families.

We choose the 35% representative dataset, consisting of 6003 domains from CATH, where sequence pairs have at most 35% amino acid sequence identity. Since ProGreSS

can not handle a large dataset, we selected 2000 domains randomly out of the 35% representative set of CATH as our dataset D. From topologies with least 8 proteins, one protein is chosen randomly as the query, resulting in a query set D_q, having 42 proteins.

Similar to the previous section, we perform three different tests to evaluate our algorithm: The *retrieval* test finds the number of correct matching structures from the same topology as the query among the top k scoring proteins, and the *classification test* tries to classify the query at the topology and class levels. The *performance test* compares the algorithms in terms of the total running time.

Since all sequence pairs have at most 35% amino acid sequence identity, it is difficult to find the homologous proteins from the database. The results of retrieval and classification tests are expected to be worse than those of the previous section.

Retrieval Test We ran the experiments using PSIST and ProGreSS to obtain the number of proteins found from the same topology for each of the 42 queries. We use the same parameters as the previous sections.

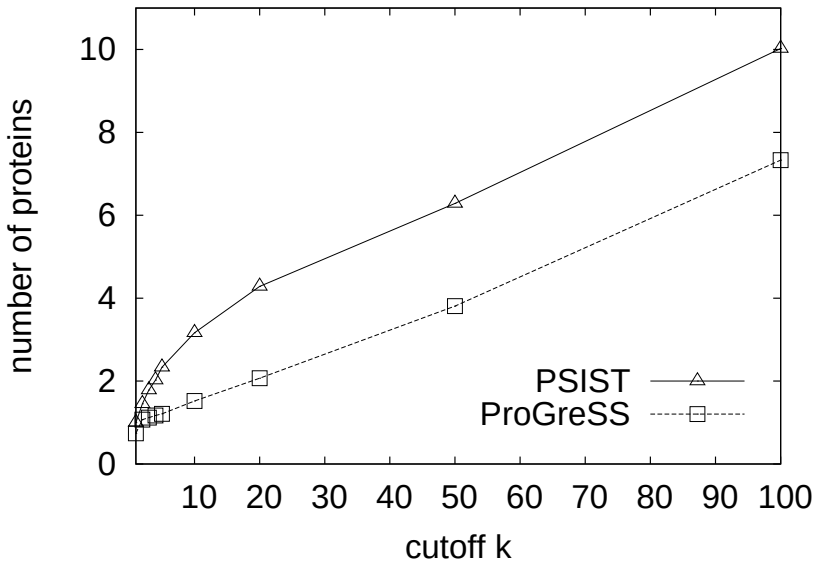

Figure 3.15: Number of proteins found from the same topology for different top-k value ($w = 3$, $b = 2$, $\epsilon = 0$ and $l = 10$).

Figure 3.15 and Table 3.5 show the number of proteins found from the same topology for different top-k cutoffs. Note that the number of correct matches is an average over all 42 CATH topologies used in our test. We find that on average PSIST returns more correct matches; for example in the top 20 results, PSIST has 4 correct matches, whereas ProGreSS returns only 2 correct matches. For the top $k = 100$, PSIST returns around 10 matches, whereas ProGreSS returns only 7.3 correct matches.

Table 3.5: Overall comparison of the number of proteins found from the same topology among the top k candidates

Algorithm	top4	top10	top50	top100
ProGreSS	1.17	1.52	3.81	7.33
PSIST	2.02	3.17	6.29	10.02

Classification Test Below we tabulate the results separately for the topology-level and class-level classification, and we report the percentage of correctly classified query proteins (out of the 42 queries). For PSIST and ProGreSS we use the best parameter settings reported in the last section.

Figure 3.16 shows the percentage of query proteins correctly classified for different length of maximal matches at the topology (TO) and class (CL) levels.

Table 3.6: CATH classification accuracy comparison at the topology (TO) and class (CL) level

Algorithm	Topology	Class
ProGreSS	7.14 %	57.1%
PSIST	50.0%	92.9 %

Table 3.6 shows the CATH classification comparison at the topology and class level respectively. ProGreSS uses both the structure and sequence features to classify the pro-

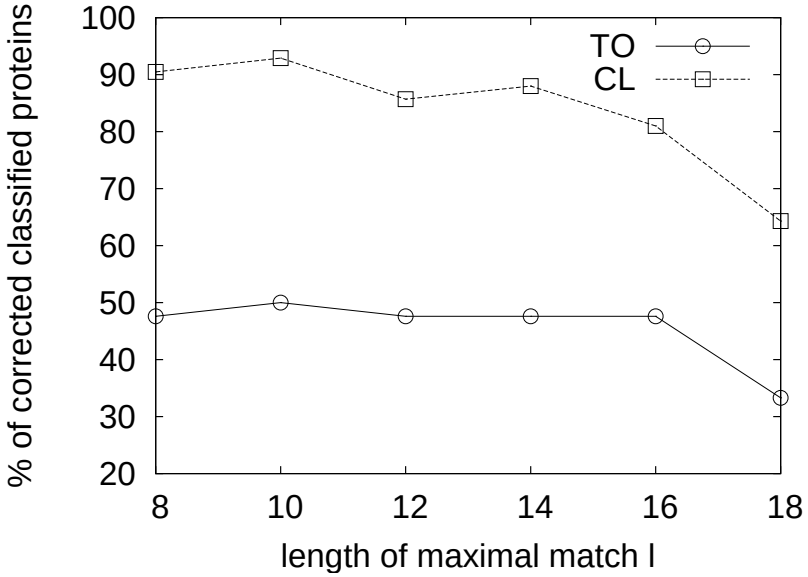

Figure 3.16: Percentage of query proteins correctly classified for different length maximal match

teins, and its accuracy is 7.14% and 57.1% at the topology and class levels. Without considering the sequence features, PSIST has much better performance than ProGreSS; its accuracy is 50.0% and 92.9% at the topology and class levels.

Table 3.7: Running time comparison

Algorithm	TO%	CL%	top10	time(s)
ProGreSS	7.14%	57.1%	1.52	1.57
PSIST-1	33.3%	64.3%	2.57	0.57
PSIST-2	47.6%	88.0%	2.93	0.95
PSIST-3	50.0%	92.9 %	3.17	2.08

Time performance Test Similar to the section 3.4.1.3, we also did three PSIST tests with different parameters. Table 3.7 shows the running time for ProGreSS and PSIST. For ProGreSS, we choose the best sequence and structure distance thresholds and set window size $w = 3$. For PSIST, we set the same parameters $w = 3$, $b = 2$, $\epsilon = 0$ for all three cases, but different length of maximal matches: $l = 18$ for the first case, $l = 14$ for

the second case and $l = 10$ for the third case. All three cases have better retrieval and classification performances compared to ProGreSS. The first case is 2.75 times faster than ProGreSS, the second case is 1.57 times faster, and the third case is the slowest, but it has the best performance.

3.4.2.2 Sequence and Structure Comparison

In this test, we also choose the same 35% representative set as our database, which has 6003 domains. However, we select the queries from CATH using a different method. We choose all of the *singletons* from the 35% representative set of CATH domains. If one domain is the only member in a homology family, it is called a singleton. If a topology has only one homology family, it is impossible to obtain homologous proteins of the singleton in that homology family, so we need to prune out these impossible singletons. After pruning, there are 370 singletons out of 6003 domains in the 35% representative set. These 370 singletons comprise the query set. For any of these singletons it is very hard to obtain similar proteins from other homologue families.

To evaluate the performance of our algorithm, we use Receiver Operating Characteristic (ROC) [22] score as our measurement. The ROC score is the area under the ROC curve, which plots true positives versus true negatives in the retrieved set of proteins. It combines measures of sensitivity and specificity. A score of 1 indicates perfect separation of positives from negatives, while a score of 0 means that none of the selected proteins are in the same topology as the query.

Two approaches are considered, one using the amino acid sequences, the other using the structures. The average ROC score for sequences was 26%, while the average ROC score for structures was 30%. Figure 3.17 shows the total number of queries whose ROC score exceeds a given ROC score threshold (on the x-axis). Not surprisingly, using the structural information leads to better retrieval quality.

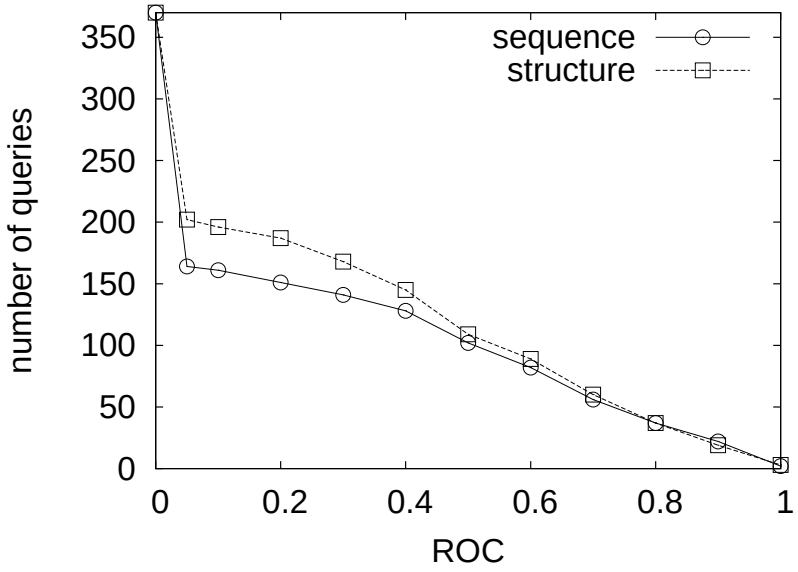

Figure 3.17: Relative performance

3.4.2.3 Indexing the SCOP Database

Since the SCOP database is curated by visual inspection it is considered to be extremely accurate. For our tests the target has all the proteins from four classes of SCOP: all α, all β, $\alpha+\beta$ and α/β. Our dataset D contains a total of $70,500$ ASTRAL SCOP 1.69 genetic domains [6]. ProGreSS ran out of memory when building the index. For PSIST, the indexing time was 3184.4 seconds and the average running time for each query was about 104.7 seconds with the parameters $w = 3$, $b = 2$, $\epsilon = 0$ and $l = 15$.

CHAPTER 4

Database indexing for local sequence-structure correlations

For the protein with unknown tertiary structure, its homologous proteins are obtained based on either the sequence similarity or its predicted structure similarity. Currently, compared with structural homology detection algorithms, most homolog detection algorithms which rely on detectable sequence similarity [1] are faster but less accurate. On the other hand, pairwise structure comparison algorithms are time-consuming.

In this chapter, we describe an algorithm for remote homolog detection using predicted local structures. Bystroff[8] presents the HMMSTR algorithm to predict the local structures from sequences. We propose a suffix tree algorithm to index the predicted structures of HMMSTR, so it can overcome the reliance on sequence and search the homologous proteins rapidly. Like PSIST, there are two steps for homolog detection: indexing the local features of predicted structure and querying the homologous proteins.

4.1 HMMSTR Structure prediction

Bystroff and Baker [7] present a new approach to predict local structure of proteins using an I-sites library of sequence-structure motifs. The I-sites library consists of a set of short sequence motifs that correlate strongly with a recurrent local structural motif in proteins. However, many I-sites motifs overlap, which results in redundant information. Furthermore, the composition of I-sites cannot capture higher order relationships such as the ordering of the motifs along the protein sequence.

HMMSTR (Hidden Markov Model for protein STRucture) [8] extends and generalizes the I-sites library. It can model the diversity of the motifs and their higher order relationships. The topology of HMMSTR is a highly branched and multi-cyclic network.

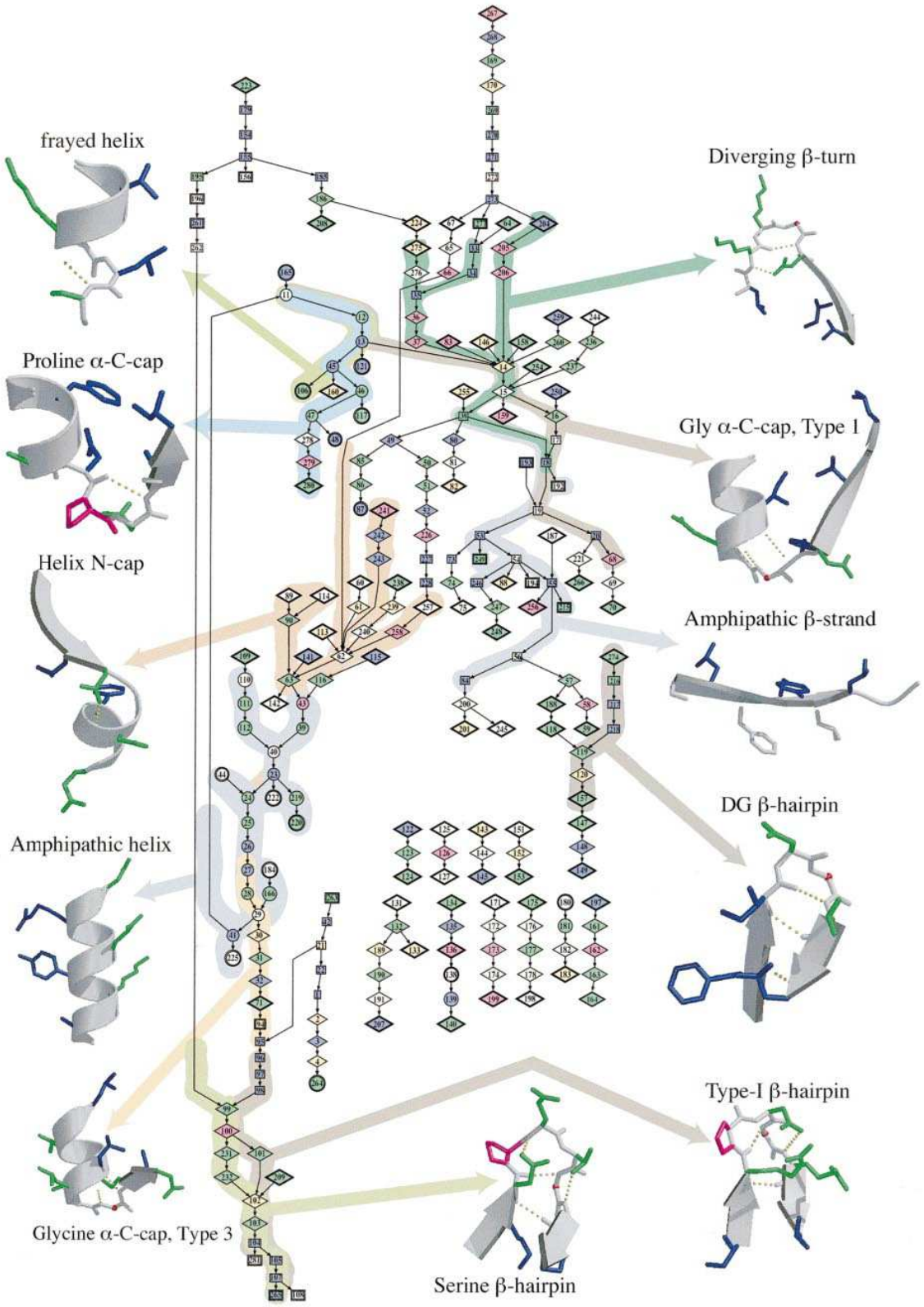

Figure 4.1: HMMSTR model [8]

Each I-sites motif was represented as a chain of Markov states, which contains information about the sequence and structure attributes of a single position in the motif. Adjacent positions are modeled by directionally linked states. A hierarchical merging of these chains of states, based on sequence and structure overlaps, results in a graph that contains almost all the motifs. The merged graph of I-sites motifs comprises a network of states connected by probabilistic transitions, or more specifically, a HMM (see figure 4.1). Each state can produce amino acid residues and structure symbols according to a probability distribution specific to that state. There are four probability distributions defined for the states in the model, b, d, r and c, which describe the probability of observing a particular amino acid, secondary structure, backbone angle region or structural context descriptor, respectively.

The database is encoded as a linear sequence of amino acid and structural observations. The amino acid sequence data consists of a "parent" amino acid sequence of known three-dimensional structure, and an amino acid profile obtained by alignments to the parent sequence [7]. The amino acid of the parent sequence is denoted by O_t, and the profile by $\{O_t^m\}(1 \leq m \leq 20)$. For the structural identifiers at each position t, the following nomenclature is used: three state secondary structure, D_t; discrete backbone angle region, R_t; and the context symbol, C_t. A sequence s of length T is given by the values of the attributes at all positions $s_t = \{O_t, \{O_t^m\}, D_t, R_t, C_t\}(1 \leq t \leq T)$

HMMSTR models protein sequences based on the notion of a path. A path is a sequence of states through the HMM, denoted $Q = q_1 q_2 \ldots q_T$. Thus, the probability of a sequence s given the model λ, $P(s|\lambda)$, is obtained by summing the relevant contributions from all possible paths Q:

$$P(s|\lambda) = \sum_{Q} \pi_{q_1} B_{q_1}(s_1) a_{q_1 q_2} B_{q_2}(s_2) \ldots a_{q_{T-1} q_T} B_{q_T}(s_T)$$

where π_{q_i} is the probability of initiating a sequence at state i, $B_{q_i}(s_t)$ is the probability

of observing s_t at state i and $a_{q_i q_j}$ represents the probability of a transition from state i to state j.

Given a HMMSTR model and a sequence, the Markov states of the sequence is predicted as follows. At first, PSI-BLAST is run against a non-redundant (NR) database to generate multiple sequence alignment. Then the multiple sequence alignment is converted to a sequence profile. Next, the profile is aligned to HMMSTR to get a probability distribution over all states at each position, giving the so-called gamma matrix. This matrix is computed by the Forward-Backward algorithm [52] and describes the probability of each HMMSTR state at each position:

$$\gamma_{pq} = P(q|s_t, \lambda)$$

for all the 281 HMMSTR states $(1 \leq q \leq 281)$ and all residues s_t $(1 \leq t \leq T$, where T is the length of the protein).

After the prediction, each position of protein sequence is converted to a 281 dimensional gamma vector. Thus, the whole protein sequence is represented by a gamma matrix, which is composed of T gamma vectors of length 281.

In addition to the gamma matrix, HMMSTR also gives a prediction of the three-state secondary structure symbol for each position in the sequence, which is obtained by summing the state-specific probability distribution multiplied by the position-specific state probability, over all states. The predicted secondary structure state is the one with the highest value in the summed distribution.

4.2 Indexing

In the previous section, the structures are predicted from the protein sequence using HMMSTR [8], and the outputs include gamma vectors and three-state secondary structure (helix, strand and turn) probabilities for each position of the protein.

There are two steps for indexing. The first is to obtain the indexable local features from the predicted structures. For each output of HMMSTR, we generate different features accordingly. The second step is to use a suffix tree to index the local features. Here we use the same suffix tree indexing algorithm as in section 3.2.3.

4.2.1 Local feature generation

4.2.1.1 Gamma matrix feature

Since the dimension of the gamma vector obtained in section 4.1 is 281, and it is difficult to index such a high dimension vector, we need to reduce the dimension first.

We cluster the gamma matrix of a protein set from the non-redundant CATH representatives. Assume there are n proteins, and that each protein p has m_p residues on average, then the total number of gamma vectors will be nm_p. They are grouped into K clusters, and each cluster is represented by a centroid.

To cluster the gamma vectors, a distance function must be defined. Hou [26] defines a similarity score of two gamma vectors u and v as their cosine:

$$cos(u, v) = \frac{u \cdot v}{\|u\|\|v\|}$$

where '$\cdot$' denotes dot product, and the dot product of two vectors is the sum of the products over all the gamma states, $u \cdot v = \sum_{i=1}^{281} u_i v_i$.

For clustering, the dot product distance between u and v is defined as

$$dist(u, v) = 1 - cos(u, v)$$

The value of $cos(u, v)$ is in the range $[0, 1]$, so the distance $dist(u, v)$ is also within $[0, 1]$.

We use the K-means clustering algorithm in our experiments. Once we have obtained the cluster centroids, the next step is to map each gamma vector to a low dimensional vector. Each vector is mapped to its nearest cluster, so there will be K symbols in

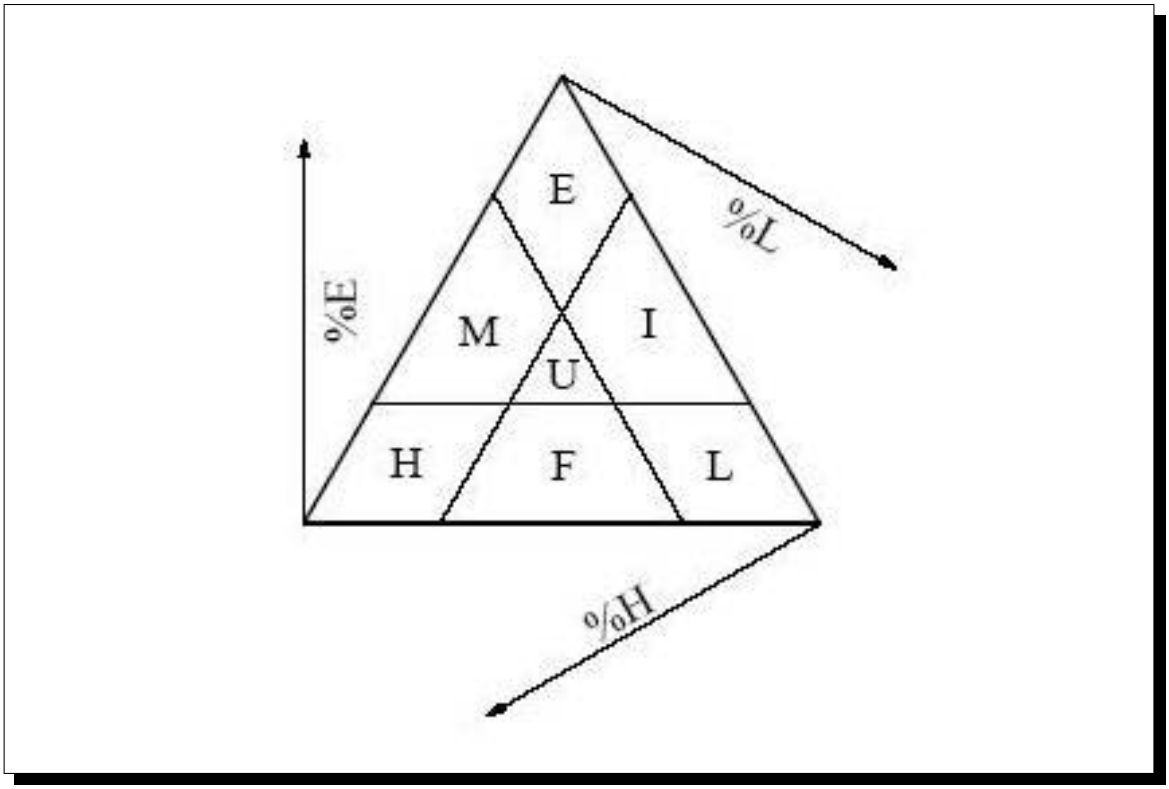

Figure 4.2: The triangular space of probabilistic 3-state predictions.

total. After clustering and mapping, each gamma vector is mapped to a K dimensional vector. Thus, each protein becomes a sequence of K dimensional vectors, and can be indexed using suffix trees in the same way as in chapter 3.

4.2.1.2 Three secondary structural states feature

We also represent proteins using the 3-state structural predictions obtained from HMMSTR. The probabilistic 3-state predictions can be represented by a triangular space, as shown in Figure 4.2, which is discretized to 7 characters. In addition to three secondary-structure states Helix(H), Strand (E) and Loop (L), we have M = not-L, I = not-H, F = not-E, and U = unknown. HMMSTR predictions are condensed to these 7 characters before constructing the suffix tree, so each protein will be represented by a sequence of 7 characters and can also be indexed by suffix trees.

4.3 Querying

To obtain the homologous proteins for a query, we search the suffix tree using a similar approach as in section 3.3. First, we search the suffix tree to retrieve all the maximal matches with the minimum length, and we then chain the maximal matches together and select the top protein with a high score. Finally we run dynamic programming for post-processing. However, there are several differences:

1. The distance function used in the NodeSearch algorithm (see figure 3.6) is Euclidean distance. However, for two mapped gamma vectors, we use the dot product distance described in section 4.2.1.1.

2. In the ranking step of section 3.3.2, the FASTA [45] algorithm is used to obtain longest chained maximal matches, whose score is the length l. Here the length of the reference protein sequence r is also considered and the new score is $s = l/r$.

3. In the post-processing step, the selected proteins are aligned to the query by running dynamic programming. We choose a different score function. Each residue is represented by a mapped vector distance, and the score between two residues is defined as the dot product distance between two vectors.

4.4 Experiments

4.4.1 Database

Remote homologs are defined as proteins that adopt the same protein fold, or topology, but whose sequences have diverged enough that conventional sequence alignments and database search methods cannot detect their similarity.

We created a dataset of proteins to test the performance of remote homology approaches. These proteins were selected from the CATH [44] hierarchy, which is based on sequence and structure similarity. CATH has four major levels of classification, namely

Class, Architecture, Topology and Homology. Homology is the lowest level; it contains either significant sequence similarity (35%) or high structural similarity and some sequence identity (20%). Protein domains that share a significant structure similarity but low sequence similarity are grouped into the same Topology. Architecture is assigned manually according to the gross arrangement of secondary structures in 3D space. At the top of the hierarchy, domains are clustered into four classes automatically by the percentage of α-helices or β-strands. Version 2.6.0 of the CATH database (Nov, 2005) contains more than 67,000 domains classified into roughly 6,000 homologous families.

We choose a set of queries from CATH as follows. To test our approach on distant homologs, we choose all of the *singletons* from the CATH domain list, where a singleton is the *sole* known structure in a family of sequence homologs (Homology family). There are 173 singletons out of 67,000 domains. If a Topology contains only one Homology family, then it is impossible to obtain a true homolog of known structure that is not the singleton itself, so we prune out these impossible singletons, to obtain the final set of 73 singletons as the query set. Each query is the only member of its Homology family, but there are at least two Homology families with known structures in the same Topology. These are difficult cases for detection. The maximum sequence identity between different Homology families is less than 35%, and this is therefore the maximum sequence identity between the query and any domain template in our test set. Our dataset of 73 queries constitutes a "hard" case for remote homology searching, since it is difficult to find the homologous domains using the sequence information alone.

4.4.2 Methods setup

We compare our approaches with PSI-BLAST[1], which is a profile-sequence method to retrieve all the homologous proteins from a database. We first describe the setup procedure of these methods.

To utilize the full power of PSI-BLAST, we choose the non-redundant (NR) se-

quence database. PSI-BLAST runs for two iterations on NR database with a large E-value threshold 100 (note that we allowed for such a large E-Value cutoff so find any possible remote homolog). The output of the second iteration was parsed and only the pdb hits were chosen.

We consider three cases/variations of our suffix tree indexing methods. Both the first two cases use the protein sequence as the input to generate the gamma matrices using HMMSTR, but they have different dimension reduction techniques. One uses the three-state probabilities of secondary structures, the other is to use k-means to cluster the 281 dimensional gamma vectors. The third case uses the protein structure/atom coordinates as the input to generate the local features (please refer to the chapter 3 for more details).

There are seven parameters used in our approaches. C is the number of clustering centroids. b is the range used to normalize the feature vectors, ϵ is the distance threshold based on the normalized feature vectors, l is the minimum length of the maximal matches, and k is the number of top scoring proteins reported, o is the opening gap penalty of dynamic programming in the post-pressing step, and e is the extension gap penalty.

4.4.3 Evaluation

For each query, a sequence of hits are obtained after searching. If the query and a hit are in the same Topology, they are homologous proteins and the hit is defined as true. To evaluate the performance of our algorithms, we use two measurements, one is Receiver Operating Characteristic (ROC) [22] score, the other is number of queries that contain true hits.

The ROC score is the area under the ROC curve, which plots true positives versus true negatives. It combines measures of sensitivity and specificity. A score of 1 indicates perfect separation of positives from negatives, while a score of 0 means that none of the selected proteins are in the same topology with the query.

4.4.4 Experimental results

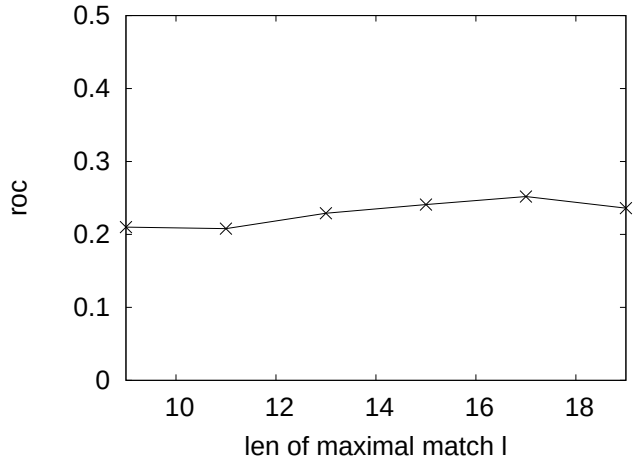

Figure 4.3: ROC score of K-Means approach for different length of maximal matches

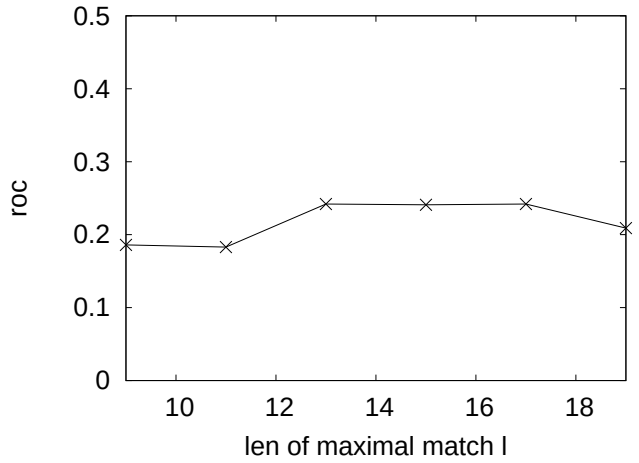

Figure 4.4: ROC score of 7 symbols approach for different length of maximal matches

Figure 4.3 and 4.4 show that the varying length of maximal match l have a similar effect on the ROC scores of the K-Means and 7 symbols approaches. For smaller maximal match length l, there can potentially be many incorrect proteins with similar match segments, but for larger l, fewer maximal matches, but correct proteins are found. Both approaches obtain the best performance when the length between 13 and 17.

Table 4.1: Remote Homology Results: PSI-BLAST is compared with two predicted structure based approaches, namely `7-symbols` which represents each residue using one of 7 symbols, and `K-Means`, which clusters the features to obtain discrete symbols using the K-Means clustering method. Also shown are the results based on the actual 3D structure, under `Structure`. The top row shows the ROC score, whereas the bottom row shows the number of true "hits" in a ranked set of results. We set the E-value = 100 for PSI-BLAST, and used $k = 7$ clusters for the K-means method. A number of other parameters were optimized (not shown) for all approaches for best performance.

Algorithms	**PSI-BLAST**	**7-symbols**	**K-means**	**Structure**
ROC	15.7%	24.9 %	24.6 %	25.3 %
#HIT	22	34	33	33

Table 4.1 shows the results of our comparison. Of the 73 test cases, PSI-BLAST was able to get a true hit for only 22 singletons, but only after using a very-high E-Value. Note that in practice, when one does not know the true answer, it would be very hard to find the true hit from the results of PSI-BLAST. In contrast, the two methods using predicted structural features return 33-34 true hits in the top 100 ranked results; they also score higher on the ROC metric. The method based on the true structure also returns 33 hits, but note that it has a slightly higher ROC score.

CHAPTER 5

Future work

Traditionally, suffix trees were mainly used in string and sequence matching, allowing either exact or approximate matches. However, they have been applied to several biological problems such as sequence indexing [29, 37] and genome alignment [13, 14]. After converting the structure to sequence by extracting the feature vectors, we have successfully applied suffix tree to attack the problem of structure similarity search and homology detection. But there is still a number of future directions.

1. In chapter 3, the feature vectors only contain structure information. However, Tang [64] claims that profile search methods based entirely on structural alignments alone are not as effective as hybrid profiles that use combined sequence-and-structure-based profiles. The feature vector of ProGreSS [5] is also a combination of sequence and structure information. Thus, one possible improvement of structure similarity search is to extract the feature vector from both the sequence and structure information and assign them different weights.

   There are many other feature representations such as contact map [73], and botwies[27]. It will be fruitful to try various representations and index them using different approaches. The ultimate goal is build a framework that can index and query the structures using different feature vectors.

2. Suffix trees are used to construct the index of the sequence and segment match in chapter 3 and 4. But the performance of suffix trees can be improved in various ways. In chapter 3, exact segment matching approaches are developed to obtain all the maximal matches, which are then assembled into longer alignments. However, for a longer matching sequence segment, it will be divided into two small maximal

segments if there is a small gap in between. Approximate searching in a suffix tree could overcome this problem. For a given query of length m, the suffix tree could be scanned down to a maximum length $k+m$, where k is the number of errors allowed for the matching segments. Currently, there are a few algorithms to match strings approximately [29], and we can build our own approximate matching algorithm specific to our problem with some biological restrictions.

Profile-sequence comparison methods utilize conserved patterns or profile of a known protein family to detect patterns of biological significance for a new sequence. Each node of the suffix tree represents a common subsequence (conserved motif) of a collection of sequences, so the concept of profile might be introduced to the suffix tree indexing.

Probabilistic suffix trees (PST) [4, 53] identify significant appearances of short segments among many protein sequences, regardless of the relative positions of these segments within the different proteins. After converting structure to sequence, we can also use PST to index protein structures and the segment start and end position information can be preserved in the tree leaves.

3. Another application of our structure indexing approach is to find good seeds for multiple structure alignment. For example, Multiprot [58], a state-of-the-art multiple structure alignment method, aims to solve the problem by detecting partial solutions; it computes the short matching segments (at least 3 residues) to obtain the transformation/alignment matrix, and then detects the largest structure cores globally between the aligned structures.

   The pseudo-code for Multiprot is given in Figure 5.1. The MultipleFragementAlignment step obtains all matching segments between the pivot protein and other proteins. It only defines 3D transformations, but it does not identify which residues are matched in 3D space, GlobalMultipleAlignment method detects the largest struc-

Input : m structures $S = M_1, ..., M_m$

for $i = 1$ *to* $m - 1$ **do**
    $M_{pivot} = M_i$
    $S' = S - M_{pivot}$
    Alignments = MultipleFragementAligment(M_{pivot}, S')
    GlobalMultipleAlignment(Alignments)

Figure 5.1: Multiprot algorithm

tural cores between the aligned structures.

Inspired by the Multiprot, the idea of PSIST can also be applied to multiple structural alignment. Basically, the algorithm is composed of the 4 steps: The first step is to compute all the matching segments (step a). For each matching segment, superimpose the matches and obtain the transformation matrix (step b), then superimpose the longest chained matching segments globally (step c). The final results are the longer alignments with smaller RMSD (step d). The details of each step are as follows.

(a) Compute the matching segments:

For protein P_i and P_j, compute all the maximal matching segments $F_i^p F_j^k(l)$ between structures using PSIST, where $F_i^p F_j^k(l)$ means a segment starts at position $p(k)$ of protein P_i (P_j) and has length l.

For each matching segment, repeat the steps b) and c) below.

(b) Align the matching segment locally: Transform the protein P_j according to the matching segment $F_i^p F_j^k(l)$:

1 Superimpose $F_i^p F_j^k(l)$.

2 Calculate $T(F_i^p F_j^k(l))$, where T is a rigid 3D transformation, and minimize distance: $RMSD(F_i^p F_j^k(l)) = \min_T RMSD(F_i^p F_j^k(l))$

3 if $(RMSD(F_i^p F_j^k(l)) > \epsilon)$ then remove $F_i^p F_j^k(l)$

4 else apply T on the whole structure F_j to obtain $T(F_j)$

(c) Align the structures globally. Using greedy algorithms to obtain the longest chained matching segments between transformed structures F_i and $T(F_j)$ and calculate the RMSD.

1 For two residues F_i^p and F_j^k, if the distance $D(F_i^p, T(F_j^k)) < \epsilon_2$, let $D(F_i^p, T(F_j^k)) = 1$, or else $D(F_i^p, T(F_j^k)) = 0$.

2 Find all the continuous 1's from the distance matrix of F_i and $T(F_j)$, then use greedy algorithm to chain the segments as in the section 3.3.2 and get the alignment.

3 Superimpose the chained segments to obtain the final RMSD.

(d) Report the results with longer alignments and smaller RMSD among the segments.

The main differences between Multiprot and our algorithm are: 1) different MultipleFragementAligment approach to obtain the matching segments (step a); 2) different GlobalMultipleAlignment approaches to do the global alignments (step c).

Preliminary results: We repeated the experiments presented in Multiprot using the same dataset. There are 10 difficult pairs of proteins they used for the structure alignment. The results of the Multiprot are taken directly from their paper [58], where the protein backbone order is preserved after alignments. Two measurements are used in the experiments: S_{al} and $RMSD$, where S_{al} is the number of aligned residues and $RMSD$ is the root mean squared distance between the aligned structures.

The open source library Bioinformatics Template Library 3.4 is used in our implementation, including the PDB file processing, superimposition and transformation.

Table 5.1: Structure alignment Comparison

Protein1(size)	**Protein2(size)**	**Multiprot** $S_{al}/RMSD$	**PSIST** $S_{al}/RMSD$
1fxi:A(96)	1ubq(76)	44/1.7	41/1.74
1ten(89)	3hhr:B(195)	81/1.3	80/1.67
3hla:B(99)	2rhe(114)	60/1.3	58/1.99
2aza:A(129)	1paz(120)	75/2.0	67/2.28
1cew:I(108)	1mol:A(94)	76/1.8	71/1.69
1cid(77)	2rhe(114)	84/1.8	76/1.82
1crl(534)	1ede(310)	161/2.3	130/2.41
2sim(391)	1nsb:A(310)	233/2.3	195/2.4
1bge:B(159)	2gmf:A(121)	78/2.5	65/2.04
1tie(166)	4fgf(124)	95/2.1	84/1.79

Table 5.1 shows that our results are very promising and competitive. Multiprot is slightly better than our algorithm. In the future, we plan to improve the performance of the pairwise alignment and test the multiple structure alignment performance of PSIST.

LITERATURE CITED

[1] Altschul,S., Madden,T., Schaffer,A., Zhang,J., Zhang,Z., Miller,W. and Lipman,D. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17), 3389-402.

[2] Aung,Z., Fu,W. and Tan, K.L. (2003) An efficient index-based protein structure database searching method. *Intl. Conf. on Database Systems for Advanced Applications (DASFAA)*, 311-318.

[3] Baeza-Yates,R. and Ribeiro-Neto,B. (1999) Modern Information Retrieval, Addison Wesley.

[4] Bejerano,G. and Yona,G. (2001) Variations on probabilistic suffix trees: statistical modeling and prediction of protein families. *Bioinformatics* 17(1):23-43

[5] Bhattacharya,A., Can,T., Kahveci, T., Singh, A.K. and Wang, Y.F.(2004) ProGreSS: Simultaneous Searching of Protein Databases by Sequence and Structure. *Pacific Symp. Bioinformatics*, 264-275.

[6] Brenner,S.E., Koehl,P. and Levitt,M. (2000) The ASTRAL compendium for sequence and structure analysis, Nucleic Acids Research, 28, pp. 254-256.

[7] Bystroff,C. and Baker,D. (1998) Prediction of local structure in proteins using a library of sequence-structure motifs. *J. Mol. Biol.* 281:565-577

[8] Bystroff,C., Thorsson,V. and Baker,D. (2000) HMMSTR: a hidden markov model for local sequence-structure correlations in proteins. *J. Mol. Biol.* 301:173-190

[9] Çamoğlu,O.,Kahveci,T., Singh,A.K. (2003) Towards index-based similarity search for protein structure databases. *IEEE Computer Society Bioinformatics Conference (CSB)*, 148-158.

[10] Can,T., Wang and Y.F.(2003) CTSS: a robust and efficient method for protein structure alignment based on local geometrical and biological features.*IEEE Computer Society Bioinformatics Conference (CSB)*, 169-179.

[11] Choi,I., Kwon,J. and Kim,S. (2004) Local feature frequency profile: A method to measure structural similarity in proteins. *PNAS 2004* 101(11): 3797-3802

[12] Clifford,R. (2005) Distributed Suffix Trees, Journal of Discrete Algorithms, 3(2-4), pp. 176-197.

[13] Delcher,A.L., Kasif,S., Fleischmann,R.D., Peterson,J., White, O. and Salzberg, S.L. (1999) Alignment of whole genomes. *Nucleic Acid Research* 27(11): 2369-2376

[14] Delcher,A.L. and Phillippy,A. and Carlton,J. and Salzberg,S.L. (2002) Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acid Research* 30(11) 2478-2483

[15] Dror,O., Benyamini,H., Nussinov,R. and Wolfson, H. (2003) MASS: Multiple structural alignment by secondary structures. *Bioinformatics*, 19(12):95–104.

[16] Durbin,R., Eddy,S. and Krogh,A. and Mitchison,G. Biological sequence analysis: probabilistic models of proteins and nucleic acids, Cambridge University Press, 1998.

[17] Eddy,S.R. (1998) Profile hidden markov models. *Bioinformatics* 14, 755-763

[18] Eidhammer,I., Jonassen I. and Taylor, W.R. (2000) Structure comparison and structure patterns. *J. of Comp. Bio.*,7(5): 685-716

[19] Eissen,S.M., Stein,B. and Potthast,M. (2005) The Suffix Tree Document Model Revisited, 5th Intl. Conference on Knowledge Management, Graz, Austria, pp. 596-603.

[20] Farach-Colton,M., Ferragina,P. and Muthukrishnan,S. (2000) On the sorting-complexity of suffix tree construction, Journal of the ACM, 47(6), pp. 987-1011.

[21] Godzik,A. (1996) The structural alignment between two proteins: is there a unique answer? *Protein Science* 5:1325-1338

[22] Gribskov,M. and Robinson,N.L. (1996) Use of receiver operating characteristic(ROC) analysis to evaluate sequence matching. *Comput. Chem.* 20:25-33

[23] Gusfield,D. (1997) Algorithms on strings, trees, and sequences: Computer science and computational biology. Cambridge University Press, New York

[24] Holm,L. and Sander, C. (1993) Protein structure comparison by alignment of distance matrices. *J. Mol. Biol*, 233: 123-138.

[25] Holm,L. and Sander,C. (1995) 3-D lookup: fast protein structure database searches at 90% reliability. *Intl. Conf. on Intelligent Systems for Molecular Biology (ISMB)*, 179-187.

[26] Hou,Y., Hsu,W., Lee,L.M. and Bystroff, C. (2004) Remote homolog detection using local sequence-structure correlations. *Proteins* 57:518-530

[27] Huang,Z.H, Zho,X. and Song,D. (2005) High Dimensional Indexing for Protein Structure Matching using bowties. *3rd Asia-Pacific Bioinformatics Conference* Singapore

[28] Hunt,E., Atkinson,M. and Irving,R. (2001) A database index to large biological sequences, Int'l Conf. on Very Large Data Bases (VLDB).

[29] Hunt,E., Atkinson,M.P. and Irving,R.W. (2003) Database indexing for large DNA and protein sequence collections. *Proc. 2003 Int. Conf. Very Large Data Bases (VLDB)* 256-271

[30] Jaakkola,T., Diekhans,M. and Haussler,D. (2000) A discriminative framework for detecting remote protein homologies. *J. Comp. Bio.* 7(1):95-114

[31] Japp,R. (2004) The top-compressed suffix tree, 21st Annual British Nat'l Conf. on Databases.

[32] Kuang,R., Ie,E., Wang,K., Siddiqi,M., Freund,Y. and Leslie,C. (2004) Profile-based string kernels for remote homology detection and motif extraction. *Computational Systems Bioinformatics* 152-160.

[33] Lamdan,Y. and Wolfson,H.J. (1988) Geometric hashing: a general and efficient model-based recognition scheme.*Intl. Conf. on Computer Vision (ICCV)*, 238-249.

[34] MacQueen,J.B. (1967) Some methods for classification and analysis of multivariate observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, University of California Press, 1:281-297

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

[36] McCreight,E.M. (1976) A space-economic suffix tree construction algorithm. *Journal of the Association for Computing machinery* 23(2):262-272

[37] Meek,C., Patel,J.M. and Kasetty,S. (2003) OASIS: An online and accurate Technique for local-alignment searches on biological sequences. *Proc. 2003 Int. Conf. Very Large Data Bases (VLDB)* 910-923

[38] Mizoguchi,K. and Go,N. (1995) Comparison of spatial arrangements of secondary structural elements in proteins. *Protein Engineering* 8:353-362

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

[40] National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov/

[41] Nussinov,R., Leibowit,N. and Wolfson,H.J. (2001) MUSTA: a general, efficient, automated method for multiple structure alignment and detection of common motifs: Application to proteins. *J. Comp. Bio.*, 8(2):93–121.

[42] Orengo,C.A., Michie,A.D., Jones,S., Jones,D.T., Swindells,M.B. and Thornton,J.M. (1997) CATH- A Hierarchic Classification of Protein Domain Structures. *Structure.* 5(8):1093-1108.

[43] Orengo,C.A. and Taylor,W.R. (1996) SSAP: Sequential structure alignment program for protein structure comparisons. *Methods in Enzymol.*, 266:617–634.

[44] Pearl,F. and etc. (2005) The CATH Domain Structure Database and related resources Gene3D and DHS provide comprehensive domain family information for genome analysis. *Nucleic Acids Research.* 33:247-251

[45] Pearson,W.R. and Lipman,D.J. (1988). Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA* 85,2444-2448.

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

[47] Phoophakdee,B. and Zaki,M.J. (2006) TRELLIS: An effective algorithm for construction disk-based suffix trees with suffix links, Tech. Report 06-03, Rensselaer Polytechnic Institute, Troy, NY.

[48] Pietrokovski,S. (1996) Searching databases of conserved sequence regions by aligning protein multiple-alignments. *Nucleic Acids Res.* 24:3836-3845

[49] Protein Data Bank, http://www.rcsb.org/pdb/

[50] Protein Information Resource, Georgetown University Medical Center. http://pir.georgetown.edu/

[51] Protein Research Foundation, http://www.prf.or.jp/en/os.html

[52] Rabiner,L. (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE 77: 257-286

[53] Ron,D., Singer,Y. and Tishby,N. (1996) The power of amnesia: learning probabilistic automata with variable memory length. *Machine learning* 25, 117-149

[54] Rost,B. (1999) Twilight zone of protein sequence alignments. *Protein Eng.*, 12(2):85–94.

[55] Sadreyev,R.I. and Grishin,N.V. (2003) COMPASS: a tool for comparison of multiple protein alignments with assessment of statistical significance *J. Mol. Biol.* 326:317-336

[56] Schaffer,A.A., Wolf,Y.I., Ponting,C.P., Koonin,E.V., Aravind,L. and Altschul,S.F. (1999) IMPALA: matching a protein sequence against a collection of PSI-BLAST-constructed position-specific score matrices. *Bioinformatics* 15(12):1000-1011

[57] Schürmann,K.B. and Stoye,J. (2003) Suffix tree construction and storage with limited main memory, Tech. Report 0946-7831, Universität Bielefeld.

[58] Shatsky,M., Nussinov,R. and Wolfson,H.J. (2004) Multiprot - a multiple protein structural alignment algorithm. *Proteins*, 56:143-156.

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

[60] Shyu,C.R., Chi,P.H., Scott,G. and Xu,D. (2004) ProteinDBS: a real-time retrieval system for protein structure comparison, Nucleic Acids Res, 1;32(Web Server issue), W572-5.

[61] Smith,F.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195-197.

[62] Söding,J. (2005) Protein homology detection by HMM-HMM comparison. *Bioinformatics* 21(7):951-960

[63] Swiss-Prot, http://us.expasy.org/sprot/

[64] Tang,C.L., Xie,L., Koh,I.Y., Posy,S., Alexov,E. and Honig,B. (2003) On the role of structural information in remote homology detection and sequence alignment: new methods using hybrid sequence profiles. *J. Mol. Biol.* 334:1043-1062

[65] Tata,S., Hankins,R. and Patel,J. (2004) Practical suffix tree construction, Int'l Conf. on Very Large Data Bases (VLDB), pp. 36-47.

[66] Täubig,H., Buchner,A. and Griebsch,J. (2004) A Method for Fast Approximate Searching of Polypeptide Structures in the PDB *Proceedings of the German Conference on Bioinformatics (GCB04)* (October 4-6 2004, Bielefeld / Germany). Lecture Notes in Informatics (LNI) P-53,65-74.

[67] Ukkonen,E. (1995) On-line construction of suffix trees. *Algorithmica* 14(3): 249-260.

[68] Wallqvist,A., Fukunishi,Y., Murphy,L.R., Fadel,A. and Levy,R.M. (2000) Iterative sequence/secondary structure search for protein homologs: comparison with amino acid sequence alignments and application to fold recognition in genome databases. *Bioinformatics* 16(11):998-1002

[69] Wikipedia, http://en.wikipedia.org/

[70] Yeh,J.S., Chen,D.Y., Chen,B.Y. and Ouhyoung,M. (2005) A web-based three-dimensional protein retrieval system by matching visual similarity, Bioinformatics 21(13), pp. 3056-3057.

[71] Yona,G. and Levitt,M. (2002) Within the twilight zone: a sensitive profile-profile comparison tool based on information theory. *J. Mol. Biol.* 315:1257-1275

[72] Yuan,X. and Bystroff,C. (2004) Non-sequential Structure-based Alignments Reveal Topology-independent Core Packing Arrangements in Proteins. *Bioinformatics, Advance Access published online on November 5, 2004*

[73] Zaki,M.J., Jin,S. and Bystroff,C., (2003) Mining residue contacts in proteins using local structure predictions, *IEEE Transactions on Systems, Man and Cybernetics*, 33(5)789-801

[74] Zamir,O. and Etzioni,O. (1998) Web Document Clustering: A Feasibility Demonstration, In Proc. of SIGIR98, Univ. of Washington, Seattle, USA.