PSIST: A Scalable Approach to Indexing Protein Structures using Suffix Trees

Feng Gao, Mohammed J. Zaki *

Department of Computer Science, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY, 12180

Abstract

Approaches for indexing proteins, and for fast and scalable searching for structures similar to a query structure have important applications such as protein structure and function prediction, protein classification and drug discovery. In this paper, we develop a new method for extracting local structural (or geometric) features from protein structures. These feature vectors are in turn converted into a set of symbols, which are then indexed using a suffix tree. For a given query, the suffix tree index can be used effectively to retrieve the maximal matches, which are then chained to obtain the local alignments. Finally, similar proteins are retrieved by their alignment score against the query. Our results show classification accuracy up to 50% and 92.9% at the topology and class level according to the CATH classification. These results outperform the best previous methods. We also show that PSIST is highly scalable due to the external suffix tree indexing approach it uses; it is able to index about 70,500 domains from SCOP in under an hour.

Key words:

Protein Structure Indexing, External Suffix Trees, Bioinformatics

1 Introduction

Proteins are composed of chains of basic building blocks called amino acids. Traditionally the problem of determining similar proteins was approached by finding the amount of similarity in their amino acid sequences. However biologists have determined that even proteins which are remotely homologous in their sequence similarities can perform surprisingly very similar functions in living organisms [37]. This fact has been attributed to the dependency of the functional role of proteins on their actual three-dimensional (3D) structure. That is, two proteins with remote sequence homology can be functionally classified as being similar if they exhibit structural homology.

Searching the growing database of protein structures for structural homologues 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 structural 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.

In this paper, we present a fast, novel protein structure indexing method called PSIST (which stands for **P**rotein **S**tructure **I**ndexing using **S**uffix **T**rees; note

* Corresponding author.
Email addresses: gaof@cs.rpi.edu (Feng Gao), zaki@cs.rpi.edu (Mohammed J. Zaki).

that a preliminary version of this paper has appeared in [15]). 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 local structural feature vectors using a sliding window along the protein 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 translational and rotational information. After normalizing the feature vectors, the protein structure is converted to a sequence (called the *structure-feature sequence* or *SF-sequence*) of discretized symbols.

We use external 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 alignments using dynamic programming. The top proteins with the highest alignment scores are finally selected. Our results shows classification accuracy up to 50.0% and 92.9% at the topology and class level according to the CATH [33,34] classification. These results outperform the best previous method. We also show that PSIST is highly scalable due to the external suffix tree indexing approach it uses; it is able to index about 70,500 domains from SCOP [30] in under an hour!

2 Related Work

2.1 Structural Similarity Search

Protein structural similarity determination can be classified into three approaches: pairwise alignment, multiple structure alignment, and database indexing.

Pair-wise structure alignment methods can be classified into three classes [13]. The first class works at the residue level [20,40]. The second class focuses on using secondary structure elements (SSEs) such as α-helices and β-strands to align two proteins approximately [26,29,32]. The third approach is to use geometric hashing, which can be applied at both the residue [25] and SSE level [21].

Previous work has also looked at multiple structure alignment. These methods are based on geometric hashing [31], or SSE information [12]. A recent method [39] 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 [44], if one seeks geometric patterns which do not follow the sequence order. Due to their time complexity, the pairwise and multiple structure alignment approaches are not suitable for searching for similarity over thousands of protein structures. Database indexing and scalable searching approaches satisfy this requirement.

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, and the other class uses SSEs to approximate the local features of the proteins.

CTSS [6] approximates the protein C_α backbone with a smooth spline with minimum curvature. 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. ProGreSS [3] 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 [9] or BLOSUM [19].

The LFF profile algorithm [7] first extracts some representative local features from the distance matrix of all the proteins, and then each protein's 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 structural similarity between two proteins is the Euclidean distance between their LFF 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 [5] uses a R^*-tree to index a nine-dimensional feature vector, a representation of all the triplet SSEs within a range. After retrieving the matching triplet pairs, a graph-based algorithm is used to compute the alignment of the matching SSE pairs. Another SSE-based method, ProtDex [2], obtains the sub-matrices of the SSE contact patterns from the distance matrix of a protein structure. 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 suffer in retrieval accuracy and as such are not very useful for small query proteins with few SSEs.

2.2 Suffix Trees

A suffix tree is a versatile data structure for substring problems [18], and it has been used for various problems such as protein sequence indexing [23,28], genome alignment [10,11] and structural motif detection [42]. Suffix trees can be constructed in $O(n)$ time and space [27,43], and thus are an effective choice for indexing sequences.

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 [14,22,41]. To address these issues, several disk-based suffix tree algorithms have been proposed in the last few years. Some of the approaches [22,24,38,41] completely abandon the use of suffix links and sacrifice the theoretically superior linear construction time in exchange for a better locality of reference. A linear algorithm to construct distributed suffix trees (DST) was proposed in [8]. 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.

3 The PSIST Approach to Protein Structure Indexing

3.1 Indexing Proteins

3.1.1 Local Feature Extraction

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. The bond lengths, bond angles and torsion angles completely define the conformation and geometry of the protein.

The 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 1). For instance, the bond length of N-C is 1.32Å ($\mathring{A}$ denotes distance in angstroms), and the bond angle between the bonds C_α-N and N-C is 123°. Torsion angles are used to describe conformations around rotatable bonds (see Figure 2). Assume four consecutive atoms are connected by the three bonds b_{i-1}, b_i and b_{i+1}. The torsion angle of b_i is defined as the smallest angle between the projections of b_{i-1} and b_{i+1} on the plane perpendicular to bond b_i. In Figure 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 structural 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 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 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). 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 defined 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 structural 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 set of feature vectors of the protein is given 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 $\cos\theta(p_i, p_j)$ gives the cosine of 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 Feature Normalization

Each structural feature vector is a combination of distances and angles, which have different measures. A normalization procedure is performed after the feature vectors are extracted. The angle θ is in the range $[0, \pi]$, so $\cos\theta \in [-1, 1]$.

For normalizing the distances, we need to know the upper-bound on the distance between the i-th and $(i+w-1)$-th residue in the protein. From Figure 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: $d(C_{\alpha 1}, C_{\alpha 2}) <= d(C_{\alpha 1}, C) + d(C, C_{\alpha 2}) = 2.453 + 1.57 = 4.023$. 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 $4.023(w-1)$, so the distance between any pair of residues within a w length window is in the range $[0, 4.023(w-1)]$.

All the distances and angles are normalized and binned into an integer within the range $[0, b-1]$. We use the equation $\lfloor \frac{d \times b}{4.023(w-1)} \rfloor$ to normalize and bin the distances and $\lfloor \frac{(\cos\theta+1)b}{2} \rfloor$ to normalize and bin the angles. Table 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 $[0, 9]$.

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 into 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 $b^{2(w-1)}$.

3.3 Generalized Suffix Tree Index Construction

After obtaining the SF-sequences for all proteins in the database, we use a generalized suffix tree (GST) as the indexing structure. A GST is a compact representation of the suffixes of multiple sequences in a single tree, and can be constructed in linear time [43]. 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 an unique key or symbol for the suffix tree construction, and map it back to the normalized vector when we need to compute the distance between two feature vectors. For instance, the three feature vectors in Table 1 could be mapped to the symbols a, b and x respectively.

Let GST be a generalized suffix tree, we use the following notation in the rest of the paper. 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 the edge label $L(E)$, $P(N)$ for the path-label of the 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.

Figure 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 vectors in Table 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. 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 $xabx$. The sequence-position identifier $(1, 0)$ of the node 7 stands for $xabxa$, the suffix of sequence S_1 that starts at position 0. Thus $sp\text{-}list(7) = \{(1, 3), (1, 0)\}$, and the sp-list for node 6 is $sp\text{-}list(6) =$

$\{(2,3),(1,3),(1,0)\}$.

3.4 Parallel/Distributed External-Memory Suffix Tree Construction

During construction of a typical in-memory suffix tree, a large amount of memory would be required to store the input strings and possibly some other bookkeeping information for large databases. This amount is normally too large for a typical computer; hence a disk-based suffix tree was selected as the method of indexing instead of an in-memory suffix tree.

In our experiments, TRELLIS [36] was applied to create the disk-based suffix tree from all of the sequences in the database. TRELLIS is an effective algorithm that builds the disk-based suffix tree based on a partitioning and merging method. It creates suffix trees for smaller substrings of the input sequence(s), and stores the suffix trees according to their common prefixes. Then, it merges the subtrees of the same prefix together, and stores the subtrees separately on disk. Let S denote the input sequence obtained by concatenating all sequences in the database. Our external-memory suffix indexing approach has three main steps:

(1) *Prefix Creation Phase:* The first step creates a list of variable-length prefixes $\{P_0, P_1, \cdots, P_{m-1}\}$. Each prefix P_i is chosen so that its frequency in the input string S does not exceed a maximum frequency threshold (determined by the main-memory limit). This also means that the number of suffixes beginning with P_i as a prefix will fit in the main-memory.

(2) *Partitioning Phase:* In the second phase, the input string S is partitioned into segments R_i (Figure 5, step a). The segment size is chosen such that

the resulting suffix tree T_{R_i} from each segment (Figure 5, step b) fits in main-memory. Each resulting suffix tree is further split into smaller subtrees T_{R_i,P_j} (Figure 5, step c), that share a common prefix P_j, which are then stored on the disk.

(3) *Merging Phase:* After processing all segments R_i, we merge all the subtrees T_{R_i,P_j} for each prefix P_j from the different partitions R_i into a merged suffix subtree T_{P_j} (Figure 5, step d). The prefixes P_j were chosen so that their suffix subtrees also fit entirely in memory. As each merged subtree T_{P_j} is constructed, it is written to disk. The complete suffix tree is simply a forest of these prefix-based subtrees (T_{P_j}).

Parallel/Distributed Suffix Tree Indexing: The idea of prefix partitioning and merging is very suitable for parallel or distributed suffix tree construction. For the prefix creation phase, let's assume initially that the set of variable-length prefixes is known. In this case, the concatenated input sequence S can be partitioned among the available processors, and each processor can obtain the local frequency of each prefix in its assigned segment (note that some overlap has to be allowed among the sequence segments to take care of boundary conditions). A summation over the processors yields the global frequencies for the set of prefixes. Since the prefix set is, in fact, not known *a priori*, the parallel prefix frequency computations can be done in multiple count-reduce iterations. In each iteration, prefixes up to a given length are counted (only those that exceed the frequency threshold in the last iteration), and a reduction is done to obtain the global frequencies.

The partitioning phase is straightforward to parallelize, since each partition is independent. Essentially, each processor builds the complete suffix tree T_{R_i} for partition R_i and splits them into the prefix-based suffix subtrees T_{R_i,P_j},

and stores them on disk. Since the partitions are all of the same size (with the exception of the last partition), a simple round-robin partition assignment scheme is sufficient to ensure good load balancing among the processors.

For the merging phase, we assign the variable-length prefixes among the processors. Each processor is responsible for merging the subtrees T_{R_i,P_j} from all the partitions $R_0, R_1, \cdots, R_{r-1}$, for a given prefix P_j. The main complication here is that prefix-based suffix subtrees for partitions assigned to other processors in the second phase, may not be available locally. Thus before the merge phase, each processor communicates its prefixed-based suffix subtrees for prefix P_j to the processor responsible for constructing the merged suffix tree T_{P_j}. Note that for the merging phase also a simple round-robin prefix assignment scheme suffices to achieve good a load balance, since each prefix yields suffix subtrees of approximately the same size.

4 Querying

Given a query Q, we first extract its feature vectors and convert it into a SF-sequence Q^s as described above. Then two phases are performed: searching and ranking. The searching phase retrieves all the matching segments/subsequences from the database within a distance threshold ϵ (on a per symbol basis), and the ranking phase ranks all the proteins by chaining the matching segments.

4.1 Searching

For a given query SF-sequence $Q^s = \{Q_1^s Q_2^s \ldots Q_n^s\}$, maximum feature distance threshold ϵ, and a minimum match length threshold l, the search algo-

rithm 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 a 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 $j = 1, 2 \dots m$, 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) For any SF-subsequence P^s of protein R^s 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, *abx* is a maximal match between the SF-sequences *xabxa* and *babxba* in Figure 4. Note that our approach differs from MUMmer genome alignment method presented in [10] which finds *exact* maximal *unique* matches between two genomes.

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. However in this approach, if there are common prefixes among the suffixes, they will be searched multiple times, leading to more comparisons than necessary. 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. This way, each common prefix is searched only once. 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 6, 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 the database node, MMS calls the NodeSearch procedure (see Figure 7) to match their labels and follow the path to find all the matching nodes.

In the NodeSearch procedure (Figure 7), 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 MMS procedure is called for two child nodes in step 3. 2) If one of the edge has a shorter label, the algorithm NodeSearch will be called recursively with the new input composed of all the edges out of the child node of the short edge (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 a character 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 matches.

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 the performance will be improved.

4.2 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 [35] as our ranking scheme. We calculate the alignment as a combination of the maximal matches with the highest score. The score of the alignment is the sum of the scores of the maximal matches minus the gap penalty. The length of a maximal match and a gap are used as the match score and gap penalty, respectively. Two maximal matches can be chained together if there is 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 next longest maximal 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. Finally all the candidates with small alignment scores are screened out and only the top similar proteins are selected.

5 Results and Discussion

To evaluate the performance of our algorithm we conduct an extensive set of experiments. The first test compares the performance of PSIST with ProGreSS [3], 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 [30], a database of proteins classified according to their structure. Our algorithm was implemented in C++ and all experiments reported below were done on a Power Mac G5 with 2.7GHz CPU, and 4GB Memory, running Darwin Kernel Version 8.0.0.

5.1 Comparison with ProGreSS

The CATH [33,34] 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 [32]. 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 compare our approach with one of the best previous indexing approaches, ProGreSS [3], using the Java-based code provided by its authors. 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.

To evaluate our algorithm we perform three different tests: 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.

5.1.1 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. 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. Based on our tuning experiments for PSIST we set $w = 3$,

$b = 2$, $\epsilon = 0$ and $l = 10$. 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 $w = 3$).

Figure 9 and Table 2 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.

5.1.2 Classification Test

In the classification test, we assume we do not know the topology or the class to which a query protein belongs. For each query we then classify it into one of the 42 CATH topologies and one of the four CATH classes (all α, all β, $\alpha + \beta$ and α/β) as 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 topology id, and a class id. The scores of the top k proteins from the same topology or class are accumulated. The query is assigned to the topology or class with the highest score. This classification approach can thus be thought of as k nearest neighbor classification. 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.

Table 3 shows the CATH classification comparison at the topology and class level respectively. 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.

5.1.3 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.

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. Table 4 shows the running time for ProGreSS and PSIST. For ProGreSS, we choose the best sequence and structure distance thresholds and set window size $w = 3$. 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.

5.2 Sequence and Structure Comparison

In this test, we 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) [17] 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 the ROC score for structures was 30%. Figure 10 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.

5.3 Indexing the SCOP Database

The SCOP database [30] classifies proteins according to a four level hierarchical classification, namely, family, super-family, fold and class. SCOP release 1.69 (from July 2005) contains a total of 25973 proteins and 70859 domains, spanning 2845 families, 1539 super-families, and 945 folds. 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 [4]. 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 $l = 15$.

6 Conclusion

In this paper, we presented a new local feature representation for protein structures. We transform the structure indexing problem into a sequence indexing problem by directly indexing the structure-feature sequences using suffix trees. The suffix trees enable rapid retrieval of maximal matching segments, which are chained into longer local structural alignments. Finally the matches are ranked according to their alignment scores. Compared to ProGreSS, our approach can index much larger databases, and at the same time it obtains higher retrieval accuracy. We also show that PSIST is highly scalable due to the distributed, and external suffix tree indexing approach it uses; it is able to index about 70,500 domains from SCOP [30] in under an hour!

Acknowledgment

This work was supported in part by NSF CAREER Award IIS-0092978, DOE Career Award DE-FG02-02ER25538, NSF grant EIA-0103708, and NSF grant EMT-0432098. We thank Benjarath Phoophakdee for her help in integrating the TRELLIS external suffix tree method with PSIST. We also thank Tolga Can, Arnab Bhattacharya and Ambuj Singh for providing us the ProGreSS code and other assistance. Finally we thank Chris Bystroff for the many helpful suggestions.

References

[1] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, D. Lipman, Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, Nucleic Acids Research, 25(17), 1997, pp. 3389-3402.

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

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

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

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

[6] T. Can, Y.F. Wang, CTSS: a robust and efficient method for protein structure alignment based on local geometrical and biological features, IEEE Computer

Society Bioinformatics Conference (CSB), 2003, pp. 169-179.

[7] I. Choi, J. Kwon, S. Kim, Local feature frequency profile: A method to measure structural similarity in proteins, Proc. Nat'l Acad. Sci. USA, 101(11), 2004, pp. 3797-3802.

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

[9] M.O. Dayhoff, R.M. Schwartz, B.C. Orcutt, A model of evolutionary change in proteins. In Atlas of Protein Sequence and Structure, M.O. Dayhoff (ed.), pp. 345-352, 1978.

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

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

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

[13] I. Eidhammer, I. Jonassen, W.R. Taylor, Structure comparison and structure patterns, Journal of Computational Biology, 7(5), 2000, pp. 685-716.

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

[15] F. Gao, M.J. Zaki, PSIST: Indexing Protein Structures using Suffix Trees, IEEE Computational Systems Bioinformatics Conference, 2005.

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

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

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

[19] S. Henikoff, J.G. Henikoff, Amino acid substitution matrices from protein blocks, Proc. Nat'l. Acad. Sci. USA, 89(22), 1992, pp. 10915-9.

[20] L. Holm, C. Sander, Protein structure comparison by alignment of distance matrices, J. Mol. Biol, 233, 1993, pp. 123-138.

[21] L. Holm, C. Sander, 3-D lookup: fast protein structure database searches at 90% reliability, Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB), 1995, pp. 179-187.

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

[23] E. Hunt, M. Atkinson, R. Irving, Database indexing for large DNA and protein sequence collections, Int'l Conf. on Very Large Data Bases (VLDB), 2002, pp. 256-271.

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

[25] Y. Lamdan, H.J. Wolfson, Geometric hashing: a general and efficient model-based recognition scheme, Int'l Conf. on Computer Vision (ICCV), 1988, pp. 238-249.

[26] T. Madej, J.F. Gibrat, S.H. Bryant, Threading a database of protein cores, Proteins, 23, 1995, pp. 356-369.

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

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

[29] K. Mizoguchi, N. Go, Comparison of spatial arrangements of secondary structural elements in proteins, Protein Engineering, 8, 1995, pp. 353-362.

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

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

[32] C.A. Orengo, W.R. Taylor, SSAP: Sequential structure alignment program for protein structure comparisons. Methods in Enzymology, 266, 1996, pp. 617-634.

[33] C. Orengo, A. Michie, S. Jones, D. Jones, M. Swindells, J. Thornton, CATH - a hierarchic classification of protein domain structures. Structure, 5(8), 1997, pp. 1093-1108.

[34] F. Pearl, I. Todd, A. Sillitoe, M. Dibley, O. Redfern, T. Lewis, C. Bennett, R. Marsden, A. Grant, D. Lee, A. Akpor, M. Maibaum, A. Harrison, T. Dallman, G. Reeves, I. Diboun, S. Addou, S. Lise, C. Johnston, A. Sillero, J. Thornton, and C. Orengo, The CATH domain structure database and related resources gene3d and DHS provide comprehensive domain family information for genome analysis, Nucleic Acids Research, 33, 2005, pp. 247-251.

[35] W.R. Pearson, D.J. Lipman, Improved tools for biological sequence comparison, Proc. Nat'l Acad. Sci. USA, 85, 1988, pp. 2444-2448.

[36] B. Phoophakdee, M. J. Zaki, TRELLIS: Genome-scale Disk-based Suffix Tree Indexing, ACM SIGMOD International Conference on Management of Data, June 2007.

[37] B. Rost, Twilight zone of protein sequence alignments, Protein Engineering, 12(2), 1999, pp. 85-94.

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

[39] M. Shatsky, R. Nussinov, H.J. Wolfson, Multiprot - a multiple protein structural alignment algorithm, Proteins, 56, 2004, pp. 143-156.

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

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

[42] H. Taubig, A. Buchner, J. Griebsch, A method for fast approximate searching of polypeptide structures in the PDB, German Conference on Bioinformatics (GCB), 2004.

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

[44] X. Yuan, C. Bystroff, Non-sequential Structure-based Alignments Reveal Topology-independent Core Packing Arrangements in Proteins, Bioinformatics, 21(7), 2005, pp. 1010-1019.

Author Biographies

Feng Gao received his B.E. degree from Shandong University of Technology in 1995 and the M.E. degree from Harbin Institute of Technology in 1997. He received his Ph.D. degree in computer science from Rensselaer Polytechnic Institute in December, 2006. His research interests are in data mining and bioinformatics.

Mohammed J. Zaki is an Associate Professor of Computer Science at RPI. He received his Ph.D. degree in computer science from the University of Rochester in 1998. His research interests focus on developing novel data mining techniques and their applications, especially for bioinformatics. He has published over 160 papers on data mining, and co-edited several books (including "Data Mining in Bioinformatics, Springer-London, 2005). He is currently an associate editor for IEEE Transactions on Knowledge and Data Engineering, action editor for Data Mining and Knowledge Discovery, and is on the edito-

rial boards of Statistical Analysis and Data Mining, Scientific Programming, International Journal of Data Mining and Bioinformatics, International Journal of Data warehousing and Mining, International Journal of Business and Systems Research, and The Open Artificial Intelligence Journal. He is a recipient of the NSF CAREER Award (2001) and DOE Early Career Award (2002). He received the ACM Recognition of Service Award in 2003, and an IEEE Certificate of Appreciation in 2005.

Tables

Table 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

Table 2
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

Table 3
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 4
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

Figures

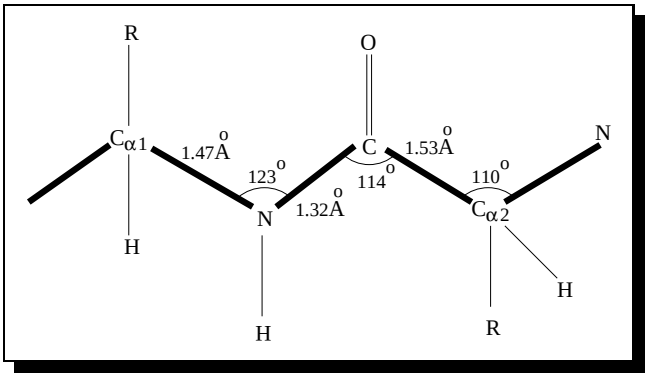

Fig. 1. Bond length and bond angles

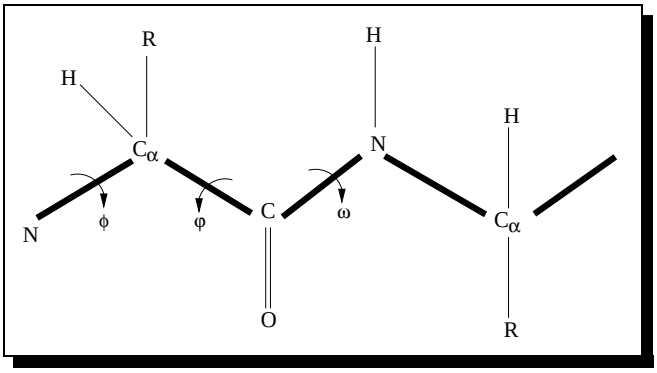

Fig. 2. Torsion angles

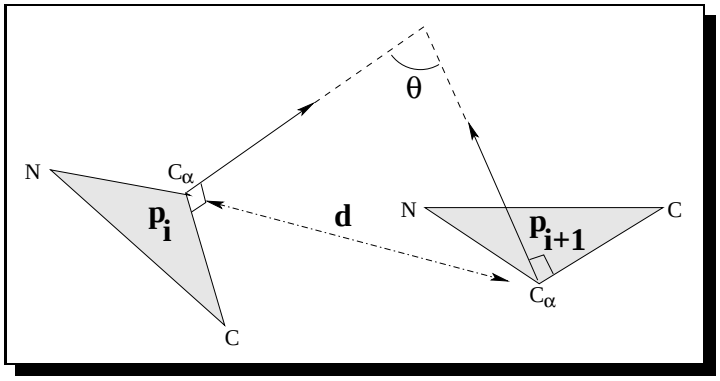

Fig. 3. The distance and angle between two residues

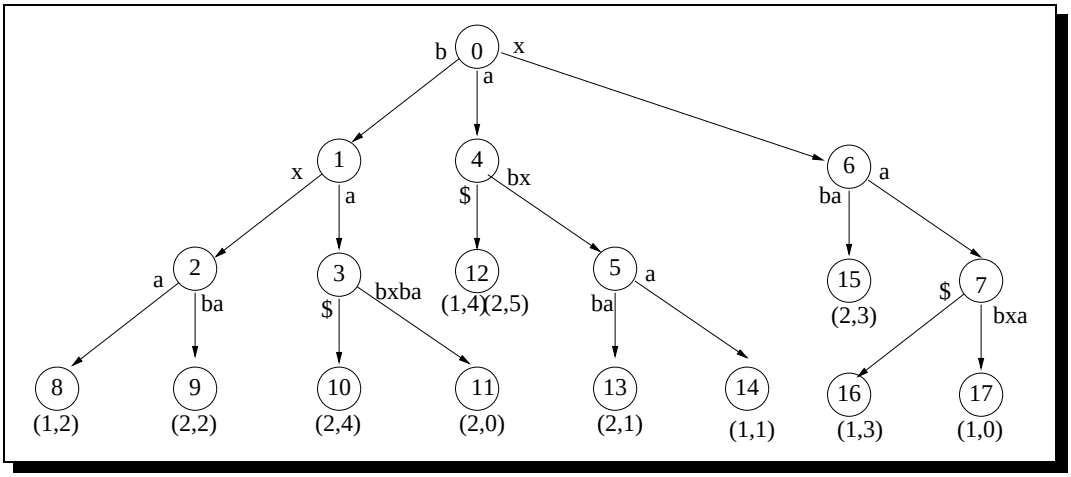

Fig. 4. GST for sequences $S_1 = xabxa$ and $S_2 = babxba$

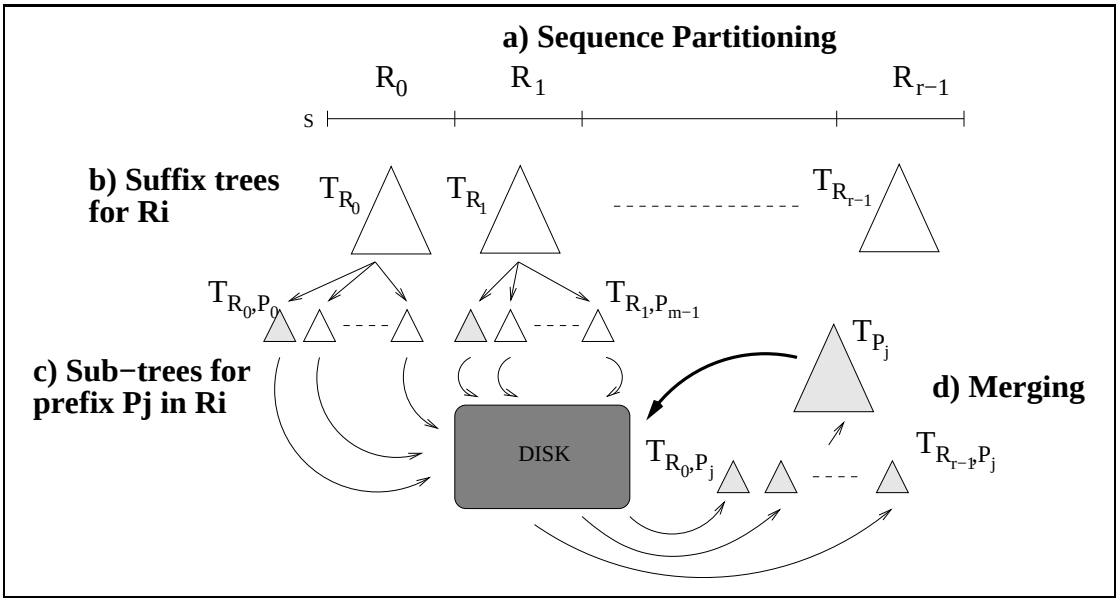

Fig. 5. Overview of External-Memory Suffix Tree

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**
    NodeSearch($E_q, 0, E_d, 0, \epsilon, l$).

Fig. 6. Maximal Match Search (MMS) Algorithm

```
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: NodeSearch(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)) > ε 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), ε, 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
              NodeSearch(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
              NodeSearch(E_q, i, E_C, 0, ε, l).
```

Fig. 7. Node Search Algorithm

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 $(x, a) \in$ *sp-list*(N_q) **do**
    foreach $(y, b) \in$ *sp-list*(N_d) **do**
      if $dist(Q_x[a-1], P_y[b-1]) > \epsilon$ **then**
        add $((x, a, a+m-1), (y, b, b+m-1))$ to $MMSet$

Fig. 8. Update Maximal Match Set Algorithm

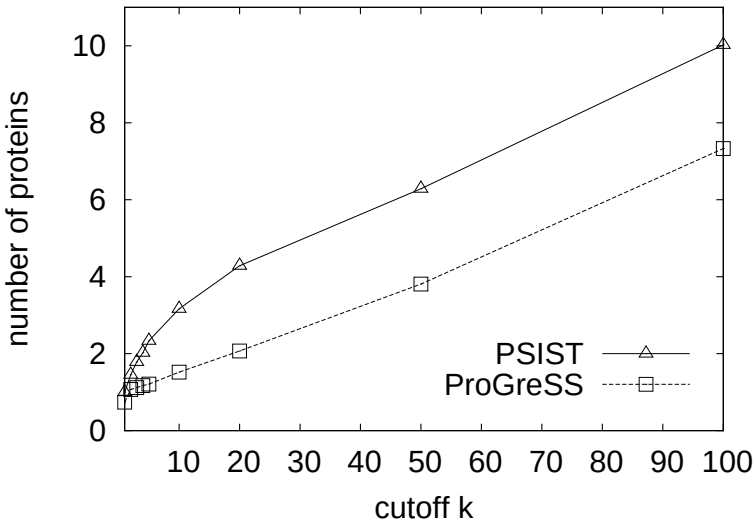

Fig. 9. Number of proteins found from the same topology for different top-k value ($w = 3$, $b = 2$, $\epsilon = 0$ and $l = 10$).

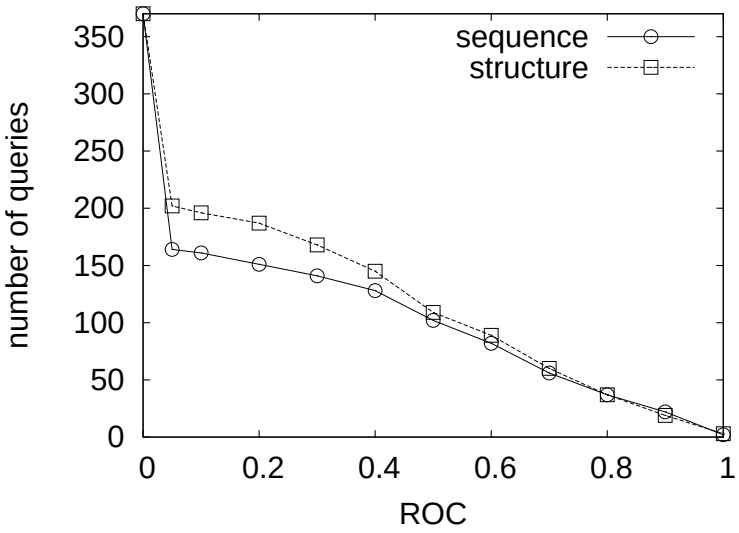

Fig. 10. Relative performance