PSIST: Indexing Protein Structures using Suffix Trees

Feng Gao and 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 developed a new method for extracting the local feature vectors of protein structures. 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. Our results shows 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.

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 [24]. This fact has been attributed to the dependency of the functional role of proteins on their actual three-dimensional (3D) structure. In view of this then it can be stated that two proteins with remote sequence homology can be functionally classified as 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.

1.1. Our Contributions

In this paper, we present a fast, novel protein indexing method called PSIST (which stands for **P**rotein **S**tructure **I**ndexing using **S**uffix **T**rees). 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 a 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. 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 suffix trees to index the protein SF-sequences. A suffix tree is a versatile data structure for substring problems [11], and they have been used for various problems such as protein sequence indexing [14][18] and genome alignment [7][8]. Suffix trees can be constructed in $O(n)$ time and space [17][29], and thus are an effective choice for indexing our 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 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.

1.2. Prior Research

Protein structural similarity determination can be classified into three approaches: pair-wise alignment, multiple structure alignments, and database indexing.

Pair-wise structure alignment methods can be classified into three classes [10]. The first class works at the residue level [12] [26]. The second class focuses on using secondary structure elements (SSEs) such as α-helices and β-strands to align two proteins approximately [16] [19] [22]. The third approach is to use geometric hashing, which can be applied at both the residue [15] and SSE level [13].

Previous work has also looked at multiple structure alignments. These methods are also based on geometric hashing [21], or SSE information [9]. A recent method [25] 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 [30], if one seeks geometric patterns which do not follow the sequence order. Due to their time complexity, the pair-wise 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 feature of the proteins.

CTSS [4] 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 or BLOSUM. Like CTSS, ProGreSS features are not localized.

The LFF profile algorithm [6] 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 [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 lack in retrieval accuracy and furthermore, are not very useful for small query proteins with few SSEs.

For a given query, the most common similarity-scoring scheme is the number of votes accumulated from the matching residues [3][4][15]. CTSS and ProGreSS further define the p-value of a protein based on the number of votes and smaller p-values imply better similarity. These scoring schemes, however, do not take into account the local similarity.

The work most relevant to our approach is PAST [15], which also uses a suffix tree to index protein structures. While PAST shares with PSIST the general idea of using a discretized alphabet to represent structural sequences, and indexing them using suffix trees, the actual details of the methods are very different. PSIST uses a different feature representation, and searches for chains of maximal matches, and most importantly is especially designed for approximate matches.

2. Indexing proteins

2.1. 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.

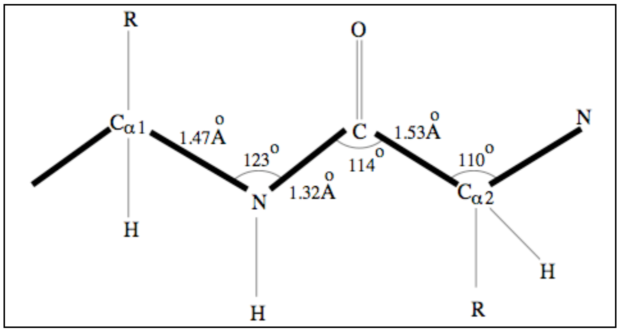


Figure 1. Bond length and bond angles

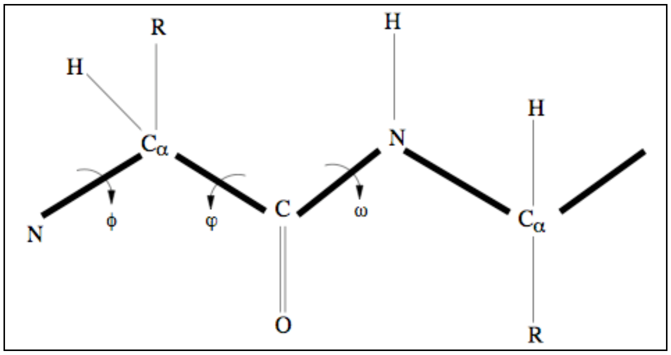


Figure 2. Torsion angles

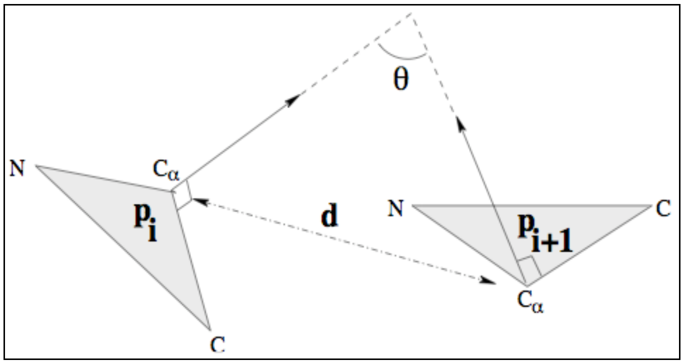


Figure 3. The distance and angle between two residues

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 **Error! Reference source not found.**). For instance, the bond length of $N\text{-}C$ is 1.32 Å (Å denotes distance in angstroms), the bond angle between $C_\alpha\text{-}N$ and $N\text{-}C$ is 123°. Torsion angles are used to describe conformations around rotatable bonds (see Figure **Error! Reference source not found.**). 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 between the projections of b_{i-1} and b_{i+1} on the plane perpendicular to bond b_i. In Figure **Error! Reference source not found.**, ϕ, ψ and ω are the torsion angles on the bond $N\text{-}C_\alpha$, $C_\alpha\text{-}C$ and $C\text{-}N$ 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 $C_\alpha\text{-}N$ bond is 1.47Å and that of the $C_\alpha\text{-}C$ bond is 1.53Å, and the angle between $C_\alpha\text{-}N$ and $C_\alpha\text{-}C$ bonds is 110°. Thus all the triangles formed by $N\text{-}C_\alpha\text{-}C$ 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 $N\text{-}C_\alpha\text{-}C$ triangles representing each residue (see Figure **Error! Reference source not found.**).

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 $N\text{-}C_\alpha\text{-}C$ triangles, is calculated between their normals having C_α as the origin. The normal of the plane define by the triangle $N\text{-}C_\alpha\text{-}C$ is given as

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

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

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

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

2.2. Normalization

Our 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 **24**, the average distance between $C_{\alpha 1}$-N atoms is $d_1 = 1.47\text{Å}$, the average distance between N-C atoms is $d_2 = 1.32\text{Å}$, 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 are 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 $[0, 4.023*(w-1)]$.

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

All the distances and angles are normalized and binned into an integer within the range $[0, b-1]$. We use the equation $d = \lfloor \frac{d*b}{4.023*(w-1)} \rfloor$ to normalize and bin the distance and $\cos\theta = \lfloor\frac{(\cos\theta+1)*b}{2}\rfloor$ to normalize and bin the angle. Table **24** 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 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 $b^{2(w-1)}$.

2.3. Generalized suffix trees construction

After obtaining the SF-sequences for all proteins in the database, we use 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 [29]. 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 compute the distance between two feature vectors. For instance, the three feature vectors in Table **24** could be mapped to the symbols a, b and x respectively.

Notation: 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.

Example: Figure **Error! Reference source not found.** 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 **Error! Reference source not found.** 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 $xabxa$, 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 $sp - list(6) = \{(2,3),(1,3),(1,0)\}$.

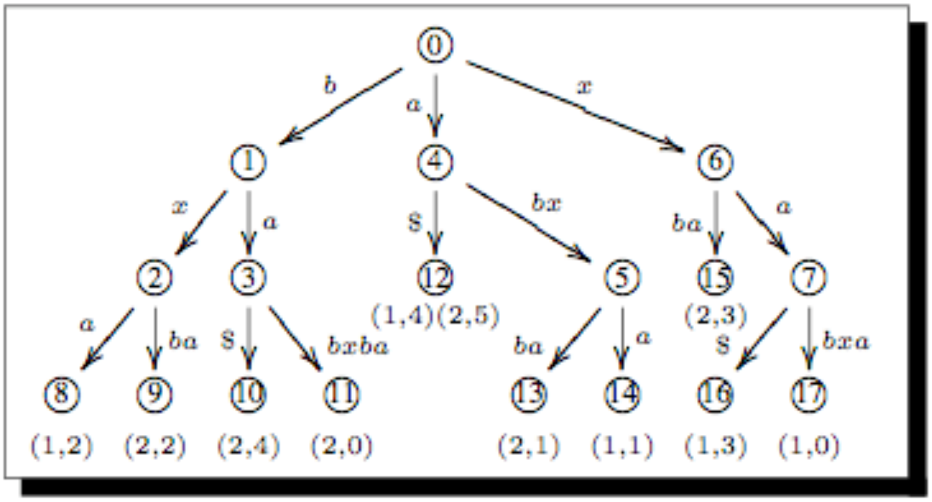


Figure 4. GST for sequences $S_1 = xabxa$ and $S_2 = babxba$

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 **Error! Reference source not found.**. 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, and the post-processing step further uses Smith-Waterman [27] approach to find the best local alignment between the query and the selected proteins.

3.1. Searching

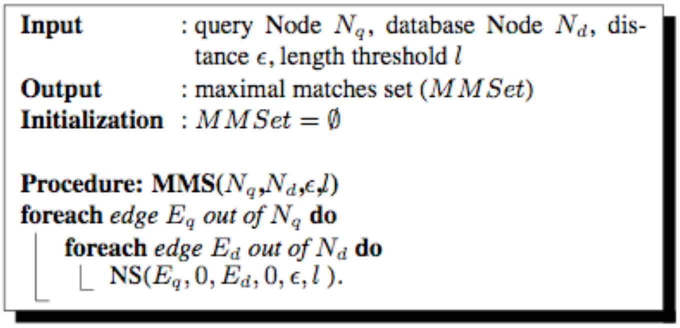
For a given query SF-sequence $Q^s = \{Q_1^s Q_2^s \dots Q_n^s\}$, maximum feature distance threshold ε, and a minimum match length threshold l, the search algorithm finds all maximal matching SF-subsequences $P^s = \{P_1^s, P_2^s \dots P_m^s\}$ that occur in both the query SF-sequence and any database 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 $dist(Q_{i+j}^s, P_j^s) < \varepsilon$, 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. Assume P^s is a SF-subsequence of protein R^s, then neither $P^s v$ nor vP^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 $babxba$ of Figure **Error! Reference source not found.**. Note that our approach differs from MUMmer genome alignment method presented in [7] 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, 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 24, 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 1) to match their labels and follow the path to find all the matching nodes.

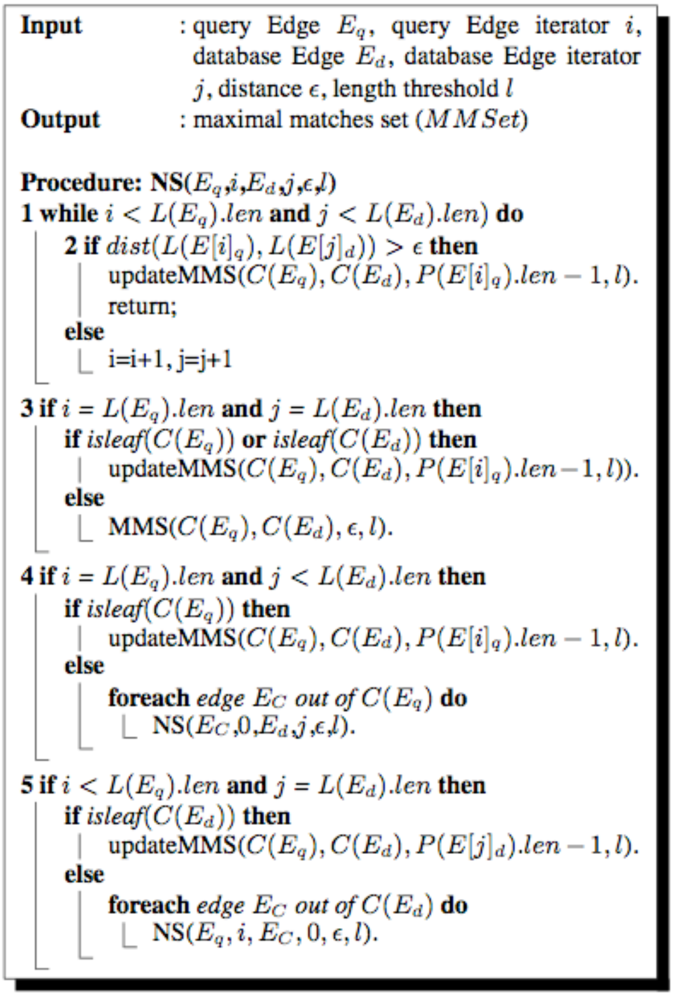


```
Input          : query Node $N_q$, database Node $N_d$, distance $\epsilon$, length threshold $l$
Output         : maximal matches set ($MMSet$)
Initialization : $MMSet = \emptyset$

Procedure: MMS($N_q$,$N_d$,$\epsilon$,$l$)
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 5. MaximalMatchesSearch algorithm

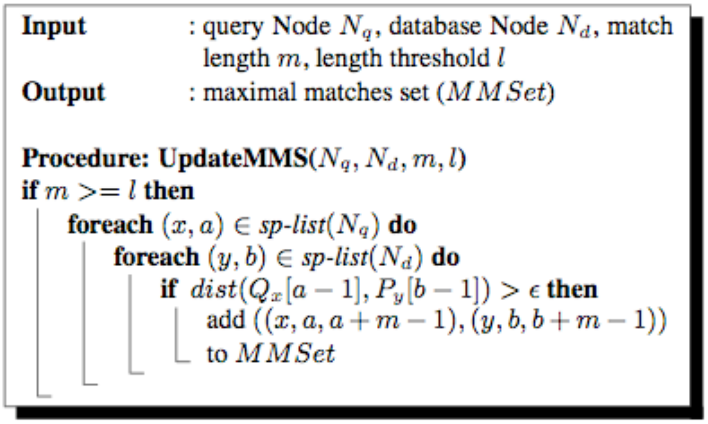
In the NodeSearch procedure, for two edges from different suffix trees, the distance between the corresponding pair of label symbols ($L(E[i]_q)$ and $L(E[j]_d)$) is computed in step 2. If the distance is larger than ε, which implies a mismatch, the procedure updates the $MMSet$ and proceeds to the next 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 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 of all the edges out of the child node of the short edge (please see step 4 and 5).



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

Procedure: NS($E_q$,$i$,$E_d$,$j$,$\epsilon$,$l$)
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,$E_d$,$j$,$\epsilon$,$l$).
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 6. NodeSearch 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 $(x, a) \in sp\text{-}list(N_q)$ do
        foreach $(y, b) \in sp\text{-}list(N_d)$ 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 $MMSet$
```

Figure 7. UpdateMaximalMatchesSet algorithm

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 the performance will be improved.

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 [23] 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 the scores of the maximal matches minus the gaps penalty. Both the score of a maximal match and a gap are their length in our algorithm. 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 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. Finally all the candidates with small alignment scores are screened out and only the top similar proteins are selected.

3.3. Post-processing

For each top protein SF-sequence with a high score selected from the database, it is aligned with the query by running Smith-Waterman [27] dynamic programming method. The similarity score between two residues is set to 1 if the distance between their normalized feature vector 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.

4. Experiments

The SCOP database [20] classifies proteins according to a four level hierarchical classification, namely, family, super-family, fold and class. 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 [3][5].

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 was implemented in C++ and all experiments reported below were done on a PC with 2.8GHz CPU and 6GB RAM, running Linux 2.6.6.

4.1. Retrieval test

We compare our approach with one of the best previous indexing approach ProGreSS [3], using the Java-based code provided by its authors. We also directly compare with a geometric hashing based [15] 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 7Å. 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, 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.

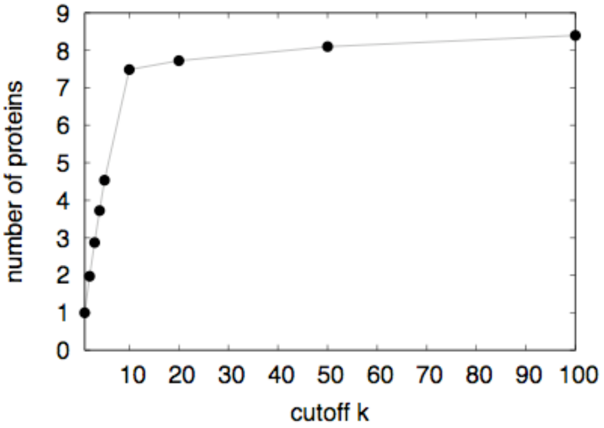


Figure 8. Number of proteins found from same superfamily for different top-k value ($w=3$, $b=10$, $\varepsilon=3$ and $l=10$).

Figure **24** 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.

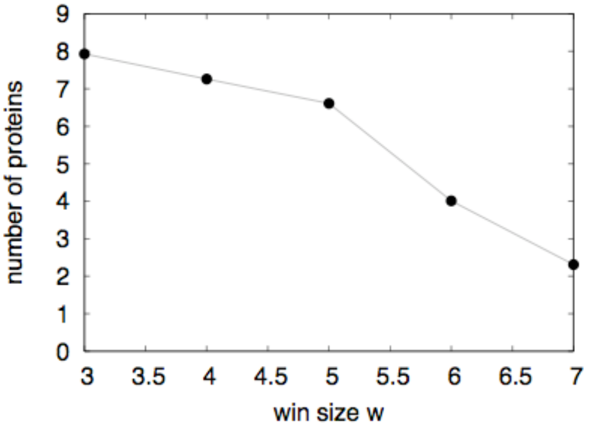


Figure 9. Number of proteins found from the same superfamily for different window sizes when ($b = 10$, $\varepsilon = 3$ and $l = 15$)

We next study the effect of varying window size w, while keeping $b = 10$, $\varepsilon = 3$ and $l = 15$. Figure **24** shows that a smaller window size of $w = 3$ yields the most number of correct matches (on average 8 correct matches out of 10), and the retrieval rate 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.

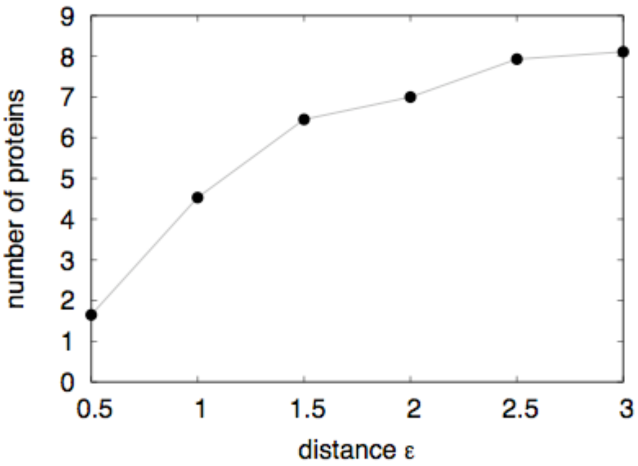


Figure 10. Number of proteins found from the same superfamily for different ε ($w = 3$, $b = 10$ and $l = 15$)

Figure 24 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 $\varepsilon = 3$ works well for PSIST, and performance tapers off for larger values.

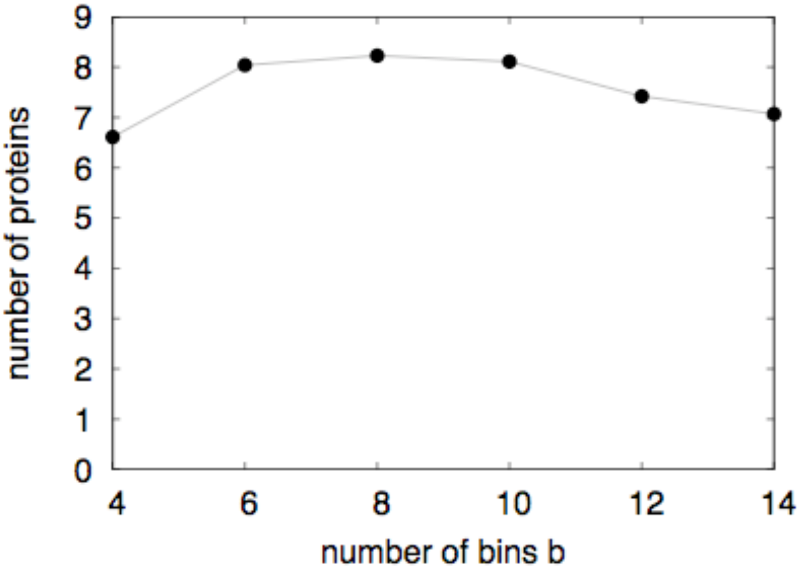


Figure 11. Number of proteins found from the same superfamily for different b ($w = 3, \varepsilon = 2.5$ and $l = 15$)

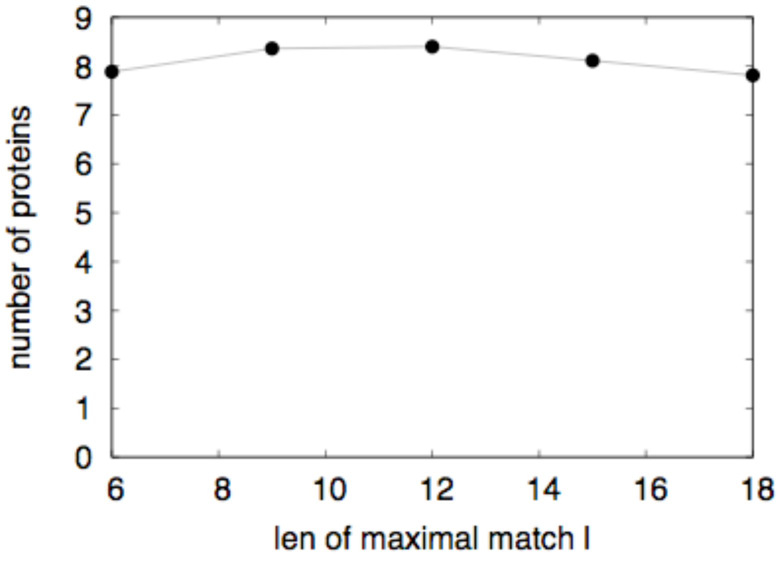


Figure 12. Number of proteins found from the same superfamily for different length of maximal matches ($w = 3$ **,** $\varepsilon = 2.5$ **and** $b = 10$ **)**

Figure **24** and **1** show that the varying normalization range b and the length of maximal match l have the 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 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 **10** 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 $\varepsilon = 0.18$, 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 $w = 3$, $b = 10$, $\varepsilon = 3$ and $l = 9$. For fair comparison, we tuned the parameter settings for ProGreSS to report its best results (we use sequence distance threshold $\varepsilon_t = 0.05$, the structure distance threshold $\varepsilon_q = 0.01$ and window size $w = 3$).

4.2. 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 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 [5] and LFF [6], 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.

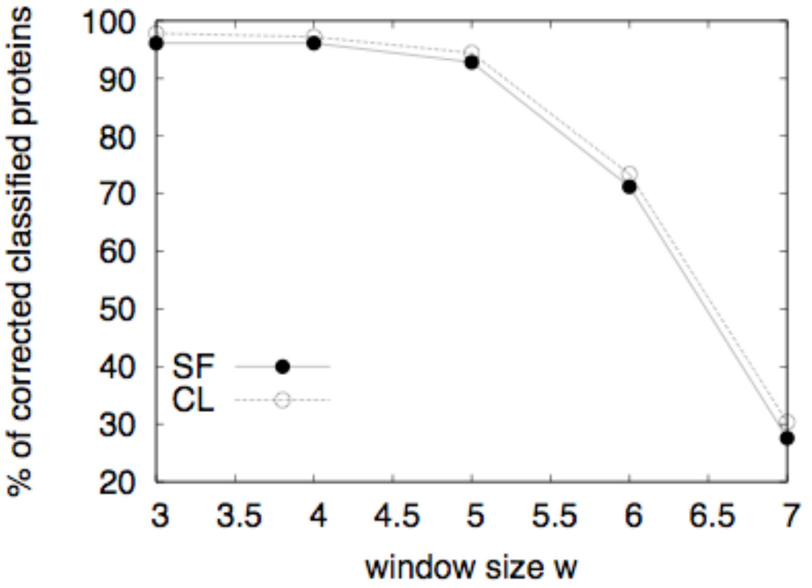


Figure 13. Percentage of query proteins correctly classified for different window sizes when ε =3

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 **Error! Reference source not found.** shows the percentage of query proteins correctly classified for different window sizes when $\varepsilon = 3$, and using $k = 3$, at the superfamily (SF) and class (CL) levels. It has a similar shape as Figure **Error! Reference source not found.**; the more the proteins found from the same superfamily, the higher the accuracy obtained.

Table 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 **Error! Reference source not found.** 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 [5] uses SSE-based features, and its accuracy for superfamily is 88%, but its class accuracy is unavailable. LFF profiles [6] 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.

4.3. 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. Assume the average number of neighbors of each reference frame is k, the complexity of our implementation of geometric hashing is $O(k*n)$. 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 $\varepsilon = 0.18$.

Table 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 **Error! Reference source not found.** 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 $w=3$, $b=2$, $\varepsilon=0$ and $l=15$ for the first case of PSIST, and it is 3.5 times faster than ProGreSS with similar retrieval and classification perform ace. The last two cases have the same parameters: $w=3$, $b=6$, $\varepsilon=2$, $l=15$, 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.

5. Conclusion

In this paper, we present a new local feature representation of protein structures and convert the structure indexing to sequence indexing. We also propose a novel use of suffix trees to find the maximal matches between structure-feature sequences and use the alignment between the query and database SF-sequences to measure the structure similarity. Compared to ProGreSS, our approach either obtains higher accuracy, or runs faster with similar classification accuracy.

Acknowledgment

We thank Tolga Can, Arnab Bhattacharya and Ambuj Singh for providing us the ProGreSS code and other assistance. We also thank Chris Bystroff and Nilanjana De for helpful suggestions. 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.

References

[1] S. Altschul, T. Madden, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped

BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389--3402, 1997.

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

[3] A. Bhattacharya, T. Can, T. Kahveci, A. Singh, and Y. Wang. Progress: Simultaneous searching of protein databases by sequence and structure. *Pacific Symp. Bioinformatics*, pages 264--275, 2004.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[21] R. Nussinov, N. Leibowit, and H. Wolfson. 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, 2001.

[22] C. Orengo and W. Taylor. SSAP: Sequential structure alignment program for protein structure comparisons. *Methods in Enzymol.*, 266:617--634, 1996.

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

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

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

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

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

[28] H. Täubig, A. Buchner, J. Griebsch: A Method for Fast Approximate Searching of Polypeptide Structures in the PDB. *Proceedings of the German Conference on*

Bioinformatics, 2004.

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

[30] X. Yuan and C. Bystroff. Non-sequential structure-based alignments reveal topology-independent core packing arrangements in proteins. *Bioinformatics, Advance Access published online*, Nov. 2004.