

# Towards Index-based Similarity Search for Protein Structure Databases \*

Orhan Çamoğlu

Tamer Kahveci

Ambuj K. Singh

Department of Computer Science

University of California, Santa Barbara, CA 93106

{orhan,tamer,ambuj}@cs.ucsb.edu

## Abstract

We propose two methods for finding similarities in protein structure databases. Our techniques extract feature vectors on triplets of SSEs (Secondary Structure Elements) of proteins. These feature vectors are then indexed using a multidimensional index structure. Our first technique considers the problem of finding proteins similar to a given query protein in a protein dataset. This technique quickly finds promising proteins using the index structure. These proteins are then aligned to the query protein using a popular pairwise alignment tool such as VAST. We also develop a novel statistical model to estimate the goodness of a match using the SSEs. Our second technique considers the problem of joining two protein datasets to find an all-to-all similarity. Experimental results show that our techniques improve the pruning time of VAST 3 to 3.5 times while keeping the sensitivity similar.

**Keywords:** Protein structures, feature vectors, indexing, dataset join

## 1 Motivation

Functional properties of proteins usually depend on structures of the proteins rather than their sequences. Predicting functional properties of proteins is needed in a number of fields such as drug design, protein classification and phylogenetics. There are many proteins which are structurally similar but their sequences are not similar at the level of amino acids. For example helical cytokines form an extended family that is undetectable by sequence comparison. This makes structural similarity more important than sequential similarity for protein classification. Detecting structural similarities of proteins using computers will be much faster and less expensive than wet lab experiments. Computational results can also be used to narrow down the possible experiments for scientists.

\* Work supported partially by NSF under grants BDI-213903 and EIA-0080134

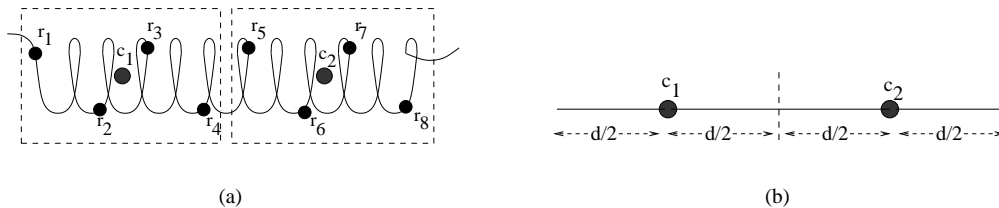
The key problem in structural alignment of proteins is to find the optimal mapping between the atoms in two molecular structures. It is not known in advance which atoms of one structure correspond to which atoms of the other structure. Searching all possible mappings requires an exponential number of comparisons in terms of the number of atoms in the compared structures. This makes an exhaustive search intractable and heuristics are frequently employed. The Root Mean Square Distance (RMSD) between the aligned atoms of two aligned structures is typically taken as a measure of the quality of the alignment. Given a mapping, the problem of optimally aligning two structures through rotation and translation so that the RMSD is minimized can be solved efficiently in time linear in the number of atoms [5].

There are three classes of algorithms for structural alignment of proteins [8]. The first class of algorithms performs structural alignment directly at the level of  $C_\alpha$  atoms. The second class of algorithms first uses the SSEs (Secondary Structure Elements) to carry out an approximate alignment and then uses the  $C_\alpha$  atoms. The final class of algorithms uses geometric hashing [23].

The simplest algorithm for structural alignment [10] uses dynamic programming to find the optimal mapping. The DALI algorithm [11] uses distance matrices to align proteins. The CE algorithm [19] performs a combinatorial extension of aligned fragment pairs. The Double Dynamic Programming algorithm [22] and Iterative Dynamic Programming algorithm [21] use two levels of dynamic programming.

Hierarchical algorithms are based on rapidly identifying mappings between small *similar* SSE fragments of two proteins. The similarity of two fragments is defined using length and angle constraints. Fragment pairs that align well form the seed for extensive atom-level alignments. This is followed by a more detailed alignment of the atoms themselves. We discuss the VAST algorithms below. Other algorithms carrying out hierarchical alignment are [3, 13, 15, 18, 20].

The VAST algorithm [14] carries out a hierarchical alignment beginning with SSEs. It begins with a bipartite graph: vertices on one side consist of pairs of SSEs from query pro-



**Figure 1.** (a) An  $\alpha$ -helix, amino acids  $r_1, \dots, r_8$ , and the center of masses  $c_1$  and  $c_2$  of two subsets of these amino acids. (b) Line approximation to the sample  $\alpha$ -helix.

tein and vertices on the other side consist of pairs of SSEs from target protein. An edge is inserted between two pairs of SSEs if they can be aligned well. A maximal clique is found in this bipartite graph; this defines the initial SSE alignment. This initial alignment is extended to  $C_\alpha$  atoms by Gibbs sampling. A nice feature of the VAST program is its ability to report on the unexpectedness of the match through a *p-value*. This is computed by considering the size of the match, the size of the proteins, and the quality of the alignment.

Geometric hashing based algorithms choose a set of reference frames from each target protein and place the other elements of the protein in a hash table, based on each reference frame. The 3-D Lookup algorithm [12] defines reference frames using SSEs. Nussinov and Wolfson [16] define reference frames based on  $C_\alpha$  atoms. The space complexity of this technique is cubic in the number of elements considered for each target protein.

As the sizes of experimentally determined [1] and theoretically estimated [2] protein structures grow, there is a need for scalable searching techniques. In this paper, we propose two novel methods for finding similarities in large protein structure datasets. For a given query (or a set of queries), our techniques can be used to find similar proteins in a target dataset quickly. We propose to extract feature vectors corresponding to triplets of SSEs. Later, an R\*-tree [6] is built on this feature space using *Minimum Bounding Rectangles (MBRs)*. Our first technique, called *PSI (Protein Structure Index)*, finds high quality seeds by aligning the SSEs that are similar to the SSEs of a given query protein. The proteins that do not have high quality seeds are removed from the target set without further consideration. We also develop a novel statistical model to compute the *p-value* of a seed. This value defines the goodness of this seed. Our second technique, called *PSI-NLJ*, finds the number of potentially similar triplet pairs by searching the feature space for join queries. Only the protein pairs that have enough similar triplets are used in the actual join operation. A high level description of the system is available in [7].

Experimental results show that PSI classified more than 88% of the superfamilies correctly. More than 98% of our results concurred with those of VAST. PSI ran 3 to 3.5 times faster than VAST's pruning step. We envision that the presented techniques will be used as a preprocessor in combination with existing structure alignment techniques that work well for modest database sizes.

The rest of the paper is organized as follows. Section 2 discusses index construction on the protein structure dataset. Section 3 discusses our search algorithm. Section 4 explains the statistical model used to evaluate the seeds. Section 5 presents the experimental results. Section 6 discusses our technique for joining datasets. We end with a brief discussion in Section 7.

## 2 A novel index for protein structures

Finding similar structures is a difficult problem. Current techniques sequentially compare the given query protein to all of the proteins in the target set to find similarities. Therefore, the cost of similarity queries of current techniques increases linearly with the size of the protein databases. This growth can be disastrous given the increase in the size of the experimental and theoretical structure databases [2]. Here, we propose to reduce the protein structure search cost by building an index on the protein structure data.

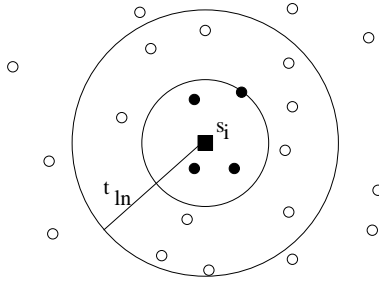
Our construction of index structure proceeds in four steps:

1. SSE approximation,
2. Triplet construction,
3. Feature vector extraction,
4. Multi-dimensional index structure construction.

Let  $\mathcal{D} = \{a_1, a_2, \dots, a_d\}$  be the set of protein structures in the dataset. We discuss these steps in more detail below.

### 2.1 SSE Approximation

Let  $a \in \mathcal{D}$  be a protein structure, where  $S_a = \{s_1, \dots, s_{n_a}\}$  is the set of SSEs of  $a$ . Let  $R_{s_i} = \{r_{i,1}, r_{i,2}, \dots, r_{i,\zeta_i}\}$  be the ordered list of residues that constitute  $s_i$ . Here,  $r_{i,k}$



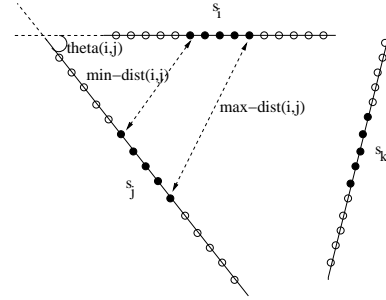
**Figure 2.** The local neighborhood set of the SSE,  $s_i$ , of a protein. The black square corresponds to the midpoint of  $s_i$ . The points represent the midpoints of the remaining SSEs of the same protein.  $t_{ln}$  is the threshold distance for the local neighborhood set. The set is then reduced to  $n = 4$  SSEs.

corresponds to the  $k^{th}$  residue in  $s_i$ . We start by splitting  $R_{s_i}$  into two equal sized lists as  $R_{s_i}^1 = \{r_{i,1}, \dots, r_{i,\zeta_i/2}\}$ , and  $R_{s_i}^2 = \{r_{i,\zeta_i/2+1}, \dots, r_{i,\zeta_i}\}$ . We define  $c_1$  and  $c_2$  as the centers of masses of the residues in  $R_{s_i}^1$  and  $R_{s_i}^2$ . A line segment approximation to  $s_i$  is achieved by extending the line segment  $[c_1, c_2]$  by half of the Euclidean distance between  $c_1$  and  $c_2$  in both directions. Figure 1(a) illustrates an  $\alpha$ -helix with eight residues, and Figure 1(b) depicts the line segment approximation to this  $\alpha$ -helix. A similar procedure is performed for  $\beta$ -strands.

## 2.2 Triplet construction

Our similarity model uses triples of SSEs, called *triplets* as the primitive similarity element to find long alignments. It is well known that the residues that are close spatially are more valuable for structural similarity [8]. In order to capture this, we construct the *local neighborhood set* of each SSE as the set of SSEs whose distances of midpoints to that SSE's midpoint are less than a predefined threshold,  $t_{ln}$ . Later, in order to limit the number of triplets, we reduce the size of the local neighborhood set to a fixed number,  $n$ , by considering only the closest of these SSEs. We use  $t_{ln} = 50 \text{ \AA}$ , and  $n = 4$  as the default values in our settings since we obtained the best results with these parameters. Figure 2 illustrates how the local neighborhood of the SSE,  $s_i$ , is selected. Here, the black square represents the midpoint of the SSE,  $s_i$ , and the black points represent the midpoints of the SSEs in the neighborhood of  $s_i$ .

Let  $V = \{v_1, v_2, v_3, v_4\}$  be the four closest SSEs to  $s_i$ . Every pair of SSEs  $v_j, v_k \in V$  forms a triplet with  $s_i$ . Therefore, the number of triplets for  $s_i$  is  $C_2^{|V|}$ , where  $C_n^m$  is defined as  $m$  choose  $n$ . Since  $|V| \leq 4$ ,  $s_i$  introduces at most  $C_2^4 = 6$  triplets. If the dataset contains  $n$  SSEs, then the total number of triplets for the entire dataset is bounded by



**Figure 3.** The extraction of feature vector for triplet  $\langle s_i, s_j, s_k \rangle$ .

$6n$ . Our experiments show that the total number of triplets for the entire PDB [1] is approximately 3.8 times the total number of SSEs.

## 2.3 Feature vector extraction

Once the triplets are determined, we construct a feature vector for each triplet for compact representation of the triplets. Such representation enables the use of index structures and similarity search.

Let  $\langle s_i, s_j, s_k \rangle$  be a triplet. We start by splitting the line segment approximation of each SSE in this triplet into three equi-sized, non-overlapping intervals. The middle interval of each SSE is then used to represent that SSE. We will explain why we chose the middle interval later in this section. For simplicity, we approximate to the middle interval by splitting each SSE into 16 equi-length intervals by placing 15 points on its line segment approximation. Later, we select the five points in the middle of each SSE to represent that SSE. Figure 3 depicts this. The black points are the points selected for each SSE in this triplet.

The pair of SSEs,  $\langle s_i, s_j \rangle$ , contributes three values to the feature vector:

- 1)  $min_{ij}$  = minimum distance between all pair of black points from  $s_i$  and  $s_j$ .
- 2)  $max_{ij}$  = maximum distance between all pair of black points from  $s_i$  and  $s_j$ .
- 3)  $\theta_{ij}$  = the angle between the line segment approximations of  $s_i$  and  $s_j$ .

Figure 3 shows these values for  $s_i$  and  $s_j$ . Since each triplet consists of three pairs, the feature vector of each triplet contains 9 values.

We choose the middle points of SSEs because their distances to the rest of the points are minimal. Therefore, they represent the SSEs better than other points. The reason that we choose five points (i.e. one-third of the line segment) can be explained intuitively as follows. If a small number

of points are used, then the feature vectors would be very sensitive to small shifts of the SSEs. If a large number of points are used, then the intervals of a feature vector would span a large interval causing large amounts of overlap. Our experimental results show that using one-third of the points is better than other schemes.

## 2.4 Index structure construction

For each triplet, we construct an object with the following fields:

- 1) The nine dimensional feature vector.
- 2) Start location on the residue list for each SSE in the triplet.
- 3) Number of residues of each SSE in the triplet.
- 4) The PDB id of the protein to which the triplet belongs.

Here, the first item contains nine float values, the second and third items contain three integer values each, and the fourth item contains five characters. The total amount of space per triplet adds up to 45 bytes per triplet. Once the objects for the feature vectors of the triplets for all the proteins in the database are created, we build an R\*-tree [6] on these objects to index them according to their feature vectors.

## 3 Our search technique

Our pruning based on SSEs consists of four steps.

*Step 1:* Similar triplets of dataset proteins and query protein are computed and stored.

*Step 2:* A *Triplet Pair Graph (TPG)* is constructed on the similar triplet pairs.

*Step 3:* A bipartite graph is constructed using the TPG. The largest matching in this graph defines the initial alignment seed at SSE level. We compute a *p-value* for each such seed at this step.

*Step 4:* The proteins that have large *p-value* are removed without further consideration. The  $C_\alpha$  alignment of the remaining proteins are determined using VAST.

We elaborate Step 1 in Section 3.1. Step 2 is discussed in Section 3.2. Finally, Step 3 is discussed in Section 3.3.

### 3.1 Finding similar triplets

Let  $q$  be the given query protein. We compute all the feature vectors of  $q$  from its SSEs as discussed in Section 2. Each feature vector corresponds to a triplet. For each feature vector, we execute a range query on the R\*-tree of the dataset proteins to find the triplet pairs, one from query protein and the other from dataset proteins, that are similar to each other.

```

Algorithm RANGE-QUERY( $t_q, R$ )
Let  $q_1, q_2, q_3$  be the SSEs in  $t_q$  in increasing length order.
Let  $\gamma_{12}, \gamma_{13}, \gamma_{23}$  be the corresponding angles.
Let  $Q$  be a queue.
1.  $\Delta(q_i, q_j) = 0.2 \cdot (\max_{q_i, q_j} - \min_{q_i, q_j})$ ; for all  $i, j$ ;
2.  $\Delta(\theta) = 10^\circ$ ;
3. QUEUE-INSERT( $Q, R$ ); /* initialize queue */
4. While  $Q \neq \emptyset$ 
    (a)  $t_d :=$  EXTRACT-QUEUE( $Q$ );
        Let  $p_1, p_2, p_3$  be the SSEs in  $t_d$  in increasing length order.
        Let  $\theta_{12}, \theta_{13}, \theta_{23}$  be the corresponding angles.
    (b) If  $([\min_{q_i, q_j} - \Delta(q_i, q_j), \max_{q_i, q_j} + \Delta(q_i, q_j)])$  overlaps
        with  $[\min_{p_i, p_j}, \max_{p_i, p_j}]$  AND  $(\gamma_{ij} \mp \Delta(\theta))$  contains  $\theta_{ij}$ 
        for all  $i, j$  then
        i. If  $t_d$  is an MBR then
            • Insert all children of  $t_d$  into  $Q$ ;
        ii. else /* i.e.  $t_d$  is a triplet. */
            • If  $(0.5 < \frac{\text{length}(p_i)}{\text{length}(q_i)} < 2)$  for all  $i$  then
                -  $S(t_q, t_d) :=$ 
                  TRIPLET-PAIR-SCORE( $t_q, t_d$ );

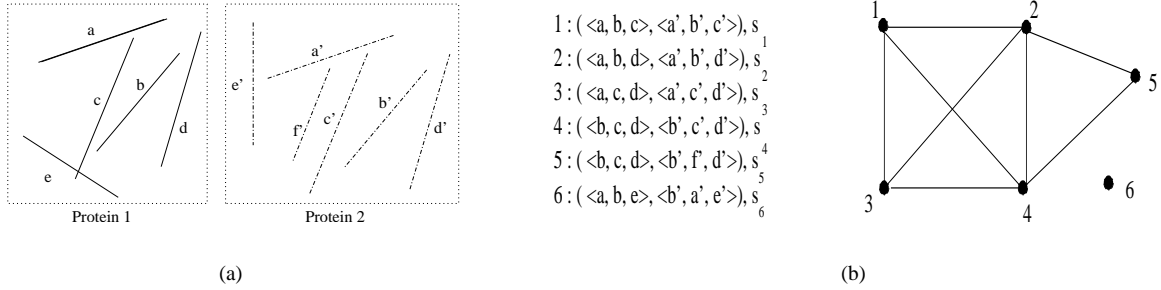
```

Figure 4. Range query algorithm.

Figure 4 shows the range query algorithm used to find similar triplets. The algorithm takes the feature vector of a query triplet,  $t_q$ , and the root node of the R\*-tree,  $R$ , as input. It starts by computing error thresholds for lengths and angles (Steps 1 and 2). A queue is initialized by inserting the root node (Step 3). While the queue contains more items, an item is extracted from the queue (Step 4.a). If the difference between the query vector and the item is less than the error thresholds in both length dimensions and angle dimensions then the item is processed (Step 4.b). If the item is an MBR, then all its children are inserted into the queue (Step 4.b.i). Otherwise, if the ratio of the lengths of the line approximations of corresponding SSEs is less than 2, then the triplet pairs are considered similar. For each similar triplet pair, a *similarity score* is calculated (Step 4.b.ii). We discard the SSE pairs if their lengths are not similar. The constants in length compatibility check affect the sensitivity and the speed of the algorithm. They are set to 0.5 and 2 based experimental results. Intuitively, these constants mean that if the length of an SSE is more than twice the length of another SSE, then they are considered as dissimilar.

The similarity score of a triplet pair  $(t_q, t_d)$ , where  $t_q$  and  $t_d$  are the query and the target triplets, is based on the distinctiveness of  $t_q$ . If  $t_q$  is similar to many triplets in the target dataset, then  $t_q$  is not distinctive. Hence, a triplet pair  $(t_q, t_d)$  has a low score. We calculate the score for  $(t_q, t_d)$  pair as:

$TRIPLET-PAIR-SCORE(t_q, t_d) = 1 - (\text{number of triplets in the dataset that align to } t_q) / (\text{total number of triplets in the dataset}).$



**Figure 5.** (a) The line segment approximations of the SSEs of two proteins. (b) The triplet pairs between the two proteins in Figure 5(a) and their scores are shown on left. The coresponding TPG is shown on right. The TPG has two connected components; one with 5 triplet pairs, the other with one.

### 3.2 Constructing Triplet Pair Graph

The range query on the  $R^*$ -tree finds similar pairs of SSE triplets. Each such pair defines a mapping of three query SSEs to three SSEs in a target protein. Mappings of larger number of SSEs can be found by merging the results of all such triplet pairs. We capture the correlation between SSE triplet pairs by building a *Triplet Pair Graph (TPG)* on similar triplet pairs. The vertices of the TPG correspond to triplet pairs and the weight of a vertex indicates the *unexpectedness* of the match.

**Definition 1** Let  $\mathcal{T}$  be the set of triplet pairs, then the Triplet Pair Graph,  $TPG(\mathcal{T}) = (V, E)$ , where  $V$  is the set of vertices and  $E$  is the set of edges such that

- 1)  $v_i \in V$  if  $v_i \in \mathcal{T}$ ,
- 2)  $e_{ij} = (v_i, v_j) \in E$  if the triplet pairs  $v_i$  and  $v_j$  have two common SSE pairs,
- 3) the weight of the vertex  $v_i$  is defined as  $TRIPLET-PAIR-SCORE(v_i, v_j)$ .

A *connected component* of the TPG corresponds to a set of triplet pairs that can be combined to find mappings of larger number of query SSEs to dataset SSEs. We run *Depth First Search (DFS)* algorithm on the TPG to find the *Largest Weight Connected Component (LWCC)*. The LWCC of the TPG is the subset of the triplet pairs that results in the highest scoring mapping of query SSEs to dataset SSEs.

Figure 5(a) depicts the line segment approximations of the SSEs of two proteins. Similar triplet pairs, their scores, and the TPG of these proteins are shown in Figure 5(b). The figure contains six triplet pairs. For example, the first triplet pair aligns the SSEs  $a$ ,  $b$ , and  $c$  to  $a'$ ,  $b'$ , and  $c'$  respectively, and the score of this alignment is  $s_1$ . The vertices of the TPG correspond to these six triplet pairs. There is an edge between first and second triplet pairs, because they both align  $a$  to  $a'$  and  $b$  to  $b'$ . As we can see there is no edge for triplet pair 6. This is because there is no other triplet pair

that share two SSE alignments with it. There are two connected components in the TPG: one of them consists of the vertices  $C_1 = \{v_1, v_2, v_3, v_4, v_5\}$ , and the other one contains only  $C_2 = \{v_6\}$ . Each of these connected components defines an alignment. However they correspond to different rigid body transformations. Therefore, both of these alignments can not occur at the same time. The scores for the connected components  $C_1$  and  $C_2$  are  $(s_1 + s_2 + s_3 + s_4 + s_5)$  and  $(s_6)$  respectively. We choose the connected component with the higher score.

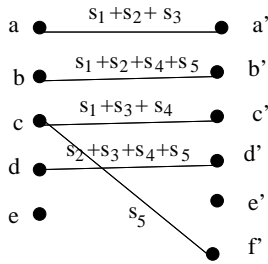
The number of nodes of the TPG is bounded by  $O(nm)$ , where  $n$  and  $m$  represent the number of target and query SSEs respectively. Since each triplet pair can introduce at most three edges to the TPG, the number of edges is also bounded by  $O(nm)$ . Therefore, the space complexity of the TPG is  $O(nm)$ . Since the time complexity of DFS is  $O(E)$ , the LWCC is found in  $O(nm)$  time.

### 3.3 Finding initial seeds

The largest weight connected component of the TPG corresponds to the most similar subset of target and the query protein's SSEs. We find an alignment of the SSEs by inspecting this subset. We start by constructing a bipartite graph on the LWCC. The bipartite graph is defined as follows:

**Definition 2** Let  $\mathcal{T}$  be a set of triplet pairs. Let  $G$  be the LWCC of the TPG of  $\mathcal{T}$ . The bi-partite graph of  $G$  is defined as  $(V_1 \cup V_2, E)$ , where

- 1)  $V_1$  is set of the query SSEs that appear in at least one of the query triplets in  $G$ ,
- 2)  $V_2$  is set of the target SSEs that appear in at least one of the target triplets in  $G$ , and
- 3) the weight of the the edge between  $q_i \in V_1$  and  $p_j \in V_2$  is the sum of the scores of the triplet pairs that align  $q_i$  to  $p_j$  in  $G$ .



**Figure 6.** The bipartite graph of the largest weight component in Figure 5(b) (assuming  $s_1 + s_2 + s_3 + s_4 + s_5 > s_6$ ). There is a conflict in the alignment of SSE  $c$ : it has an edge to both  $c'$  and  $f'$ . The edge with the largest weight is chosen to solve the conflict.

Unlike TPG, the bipartite graph consists of two disjoint vertex sets. The vertices in one set correspond to the query SSEs in the LWCC. The vertices in the other set correspond to the target SSEs in the LWCC. The weight of an edge indicates the quality of the alignment of the corresponding pair of vertices. We run a largest weight bipartite graph matching algorithm [9] on the final bipartite graph to find a mapping of the vertices in the two sets that maximize the sum of edge weights. The resulting mapping defines a seed for each target protein. Finally, we align the line segment approximations of the SSEs that constitute the seed to find their RMSD.

Figure 6 shows the bipartite graph of largest weight component of the TPG in Figure 5(b) (assuming  $s_1 + s_2 + s_3 + s_4 + s_5 > s_6$ ). Here, the vertices correspond to the SSEs in Figure 5(a). The weight of an edge is equal to the sum of the scores of the triple pairs that align those SSEs. For example, the weight of the edge between  $a$  and  $a'$  is  $s_1 + s_2 + s_3$  since first, second, and third triplet pairs align  $a$  to  $a'$ . This bipartite graph maps  $a$  to  $a'$ ,  $b$  to  $b'$ , and  $d$  to  $d'$ . We can see that  $c$  has an edge to both  $c'$  and  $f'$ . We choose the edge with largest weight to solve this conflict. For example, if  $s_1 + s_3 + s_4 > s_5$ , then we map  $c$  to  $c'$ . The SSEs  $e$  and  $e'$  are not mapped to any SSE since there is no edge for their vertices.

## 4 Evaluating seeds

So far we have discussed how the seeds are constructed using the index structure. Each seed defines an alignment of the query protein to a target protein in the feature space. In this section, we develop a statistical model and propose a formula to calculate the  $p$ -value of a seed. The  $p$ -value of a seed corresponds to the probability of having a seed at least as good as the given one in a randomly distributed space. Therefore, small  $p$ -values correspond to *unexpected*

matches. We eliminate the target proteins that have seeds with high  $p$ -value, and consider only the remaining target proteins for  $C_\alpha$  alignment. Next we discuss the computation of  $p$ -values.

Each seed is represented with the number of  $\alpha$ -helices and  $\beta$ -strands in query and target proteins, the number of SSEs aligned of each type, and the RMSD between the aligned SSEs. We define the  $p$ -value of a seed below.

**Definition 3** Assume that two proteins have  $a_1$  and  $a_2$   $\alpha$ -helices, and  $b_1$  and  $b_2$   $\beta$ -strands respectively. Let  $\psi$  be the seed with  $a$   $\alpha$  matches and  $b$   $\beta$  matches having an RMSD of  $d$ , then the  $p$ -value of  $\psi$  is defined as

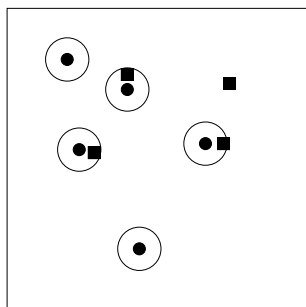
$p\text{-value}(\psi) =$  The probability that a random alignment of these two proteins contains at least  $a$   $\alpha$ -helices and at least  $b$   $\beta$ -strands within an RMSD of at most  $d$ .

Consider an example seed with five  $\alpha$ -helices in query protein and four  $\alpha$ -helices in target protein. For simplicity, each SSE is represented by a point in 3-dimensional Euclidean space. Assume that three  $\alpha$ -helices are aligned with  $\text{RMSD} = d$ . We assume that the distance between any two aligned SSEs is not greater than the RMSD of the seed. Figure 7 illustrates this example in 2-D. Here, the black circles represent query SSEs, and the black rectangles represent target SSEs. The circles around the query SSEs represent the  $d$ -distance region around query SSEs. We compute the  $p$ -value using the following observation: If a target SSE is aligned to a query SSE, then it must be located in the sphere centered at that query SSE, with radius  $d$ . Only three SSEs are aligned in Figure 7.

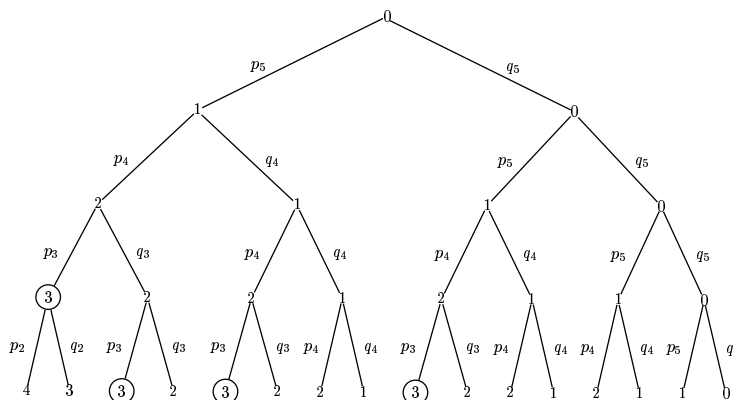
Let the number of  $\alpha$ -helices in two proteins be  $a_1$ ,  $a_2$ , then we build a binary probability tree of height  $a_2$ . Figure 8 depicts a probability tree for  $a_1 = 5$ , and  $a_2 = 4$ . The nodes at the  $k^{\text{th}}$  level show the contribution of the  $k^{\text{th}}$  SSE to the alignment after the contributions of the first  $k-1$  SSEs are computed. For example, the numbers 1 and 0 at the first level correspond respectively to the cases when the first SSE of the target protein is aligned to (or not aligned to) a query proteins SSE. The weights on the edges show the probability of the event for the child node. The value  $p_i$  is the probability that a randomly selected point is within one of the  $i$  spheres where each sphere has radius equal to given RMSD value (i.e. the probability of success.). Therefore,  $p_1$  is equal to the ratio of the volume of a sphere to the volume of the whole search space. For simplicity, we assume that spheres are non-overlapping. Hence,  $p_i = i \cdot p_1$ . The value of  $q_i$ , the probability of failure, is equal to  $1 - p_i$ .

The probability corresponding to a node in the tree can be calculated as the multiplication of the weights of all the edges on the path from the root node to that node. For example, probability of the leftmost leaf node is  $p_5 \cdot p_4 \cdot p_3 \cdot p_2$ .

The probability of having an alignment of at least  $k$  SSEs for the given proteins is calculated as the sum of the proba-



**Figure 7.** Illustration of a sample seed in 2-D. The big box represents the whole search space. Black circles represent query SSEs. Black rectangles represent target SSEs. Here, only three SSEs are aligned. The circles around query SSEs have a radius equal to the RMSD of this alignment.



**Figure 8.** The probability tree for the alignment of a query protein with five  $\alpha$ -helices and a target protein with four  $\alpha$ -helices. The probabilities of the circled nodes are accumulated to find the probability of aligning three  $\alpha$ -helices.

bilities of the nodes with value  $k$ , and which do not have an ancestor of value  $k$ . For example, in Figure 8, the probability of aligning at least three  $\alpha$ -helices is the sum of the probabilities of the nodes in circle. All the values in the probability tree can be calculated simultaneously using a dynamic programming strategy.

The p-value of a seed can be calculated as the multiplication of the probabilities of the alignment of  $\alpha$ -helices and  $\beta$ -strands.

## 5 Experimental Evaluation

We used single domain chains as the target dataset in our experiments. We created a dataset  $D_{SDC}$  of all the protein chains in PDB that contain only one domain according to VAST and SCOP classifications. We only considered proteins that are members of one of the following SCOP classes: all  $\alpha$ , all  $\beta$ ,  $\alpha+\beta$  and  $\alpha/\beta$ . In the end, the dataset  $D_{SDC}$  contained 12138 protein chains. We identified the superfamilies (according to SCOP classification) that have at least 10 representatives in  $D_{SDC}$ . There are 180 such superfamilies. Another set  $D_{SF}$  of size 1800 is created by including 10 proteins from each of these superfamilies. We also identified all folds that have at least 10 representatives in  $D_{SDC}$ , and formed another set,  $D_{CF}$ , by choosing 10 proteins from each of these 138 folds. The query set,  $D_Q$ , used in our tests is formed by choosing a random chain from each of the 180 superfamilies in  $D_{SF}$ .  $D_Q$  is large enough to sample  $D_{SDC}$  since it contains one protein from each superfamily. These 180 proteins in  $D_Q$  are listed in Table 1. We ran a number of experiments on these sets to test the quality and the performance of our techniques. The tests

are run on a computer with two AMD Athlon MP 1600+ processors with 1 GB of RAM, running Linux 2.4.19.

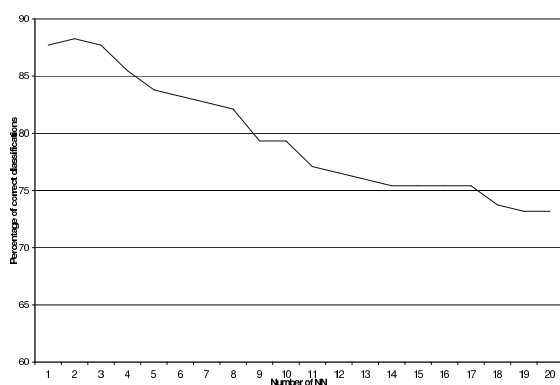
### 5.1 Quality comparison

Our first experiment set inspects the quality of the seeds found using the feature vectors by testing the correctness of the classification. We classify a given query protein into one of the superfamilies using the  $k$  best seeds as follows. The logarithms of the p-values of the seeds of the results in each superfamily are accumulated. The query protein is classified as the superfamily that has the largest magnitude of this sum. Figure 9 shows the percentage of query proteins correctly classified for different number of nearest neighbors, using  $D_{SF}$ . As can be seen from the figure, more than 86% of the proteins are classified correctly using the first two neighbors. The quality increases up to 88% for 3-NN (3-Nearest Neighbor), but the percentage drops for larger number of results. Even for 20-NN, more than 76% of the proteins are classified correctly. Actually, the drop after some point is as expected. This is because the target dataset  $D_{SF}$  contains only 9 proteins that are in the same superfamily as the query protein. Therefore, even when all the 9 proteins in the same superfamily are returned in the answer set, the remaining 11 proteins will belong to the other super families. Also the dataset may contain proteins that are structurally similar to the query protein, but belong to a different superfamily. We conclude that, our method classifies the proteins accurately using the first few results (e.g. top 3 results).

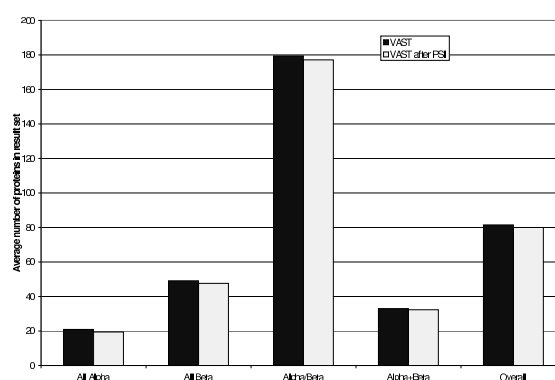
Ankerst et al [4] considered the similar problem of protein classification. Their accuracy is similar to ours. How-

**Table 1.** The PDB ids of query proteins used in our experiments.

Query ids	Queries									
1-10	1lh1	1c2n	1ret	2bby	1wjc-A	2spz-A	2lef-A	1b67-A	1rcp-B	1i4z-B
11-20	1fr0-A	1dps-I	2cyk	1unk-B	1adr	1tn4	1mj2-B	1fk1-A	1ihf-B	1b4f-B
21-30	1qck-A	2ygs-A	1hg4-B	1kvw	1i77-A	1wiu	1egj-A	1ej8-A	2msp-A	1do6-A
31-40	1g43-A	1hu8-B	1g1o-D	3pcn-N	1bqk	1gmi-A	1cov-3	1hdf-A	1shs-A	1slu-A
41-50	1bd9-A	1cx1-A	1jh5-C	1dmz-A	7cel	4hck	2vub-A	2pdz-A	1i4k-S	1pto-E
51-60	1vqi	1mjx-B	1gus-C	7ilb	1ba7-B	1inc	3hvp	1i7a-A	1ief-B	1acd
61-70	2iza	1dyw-A	1bnu	1bwu-P	1hg8-A	1thj-C	1cax-F	1hjj-A	1qaw-B	1a6x
71-80	2f3g-B	1kdf	1a5l-B	1f39-A	1hg3-D	1cw2-A	1g4s-A	1d3g-A	1az2	2xyl
81-90	1ez2-B	1fxq-B	1dxf-A	1xic	1ptd	1dhr	1dkd-C	1b2s-D	1tyl-L	1c2y-P
91-100	1f1j-B	2chf	1fh	1xze	1d0i-C	1i7s-D	1gn8-A	1cd5-A	1dts	1jh8-A
101-110	1sud	5cev-A	1qca	1jf8-A	1ypt-A	1aiu	1kc6-B	1a5v	1vfn	1a2z-C
111-120	8cpa	1jlj-B	2hpa-B	1upu-D	1bhq-2	1kpg-A	1h6j-B	1din	1mas-A	1drf
121-130	1rk2-D	1jdi-A	7icd	1qui	4mt	205l	1au0	1bxi-B	1rbg	1e1s-A
131-140	1ejr-A	1qg7-B	1azq-A	3rhv	1lfd-C	1doy	1c78-A	1igd	1gd3-A	1e3v-A
141-150	1ayz-B	2emd	1fkg	1jc4-A	1eyp-A	2ci2-I	1ec6-B	1frk	2nck-R	1fj7-A
151-160	1f9f-B	1fe4-B	1rcx-S	1dch-C	1xxb-F	2cht-E	1ott-D	1icr-B	4aig	1bkl
161-170	2hpr	1b9l-A	1i1d-A	1hqz-8	1f1l	1byw-A	1ga7-A	1b5m	1f5c-B	1qmr-A
171-179	1f7l-A	1g3i-R	1aha	1prt-G	1bnl-A	1fzd-B	1gu9-C	1jya-A	1is8-K	1lep-E



**Figure 9.** The percentage of proteins correctly classified using seed for different number of NN.



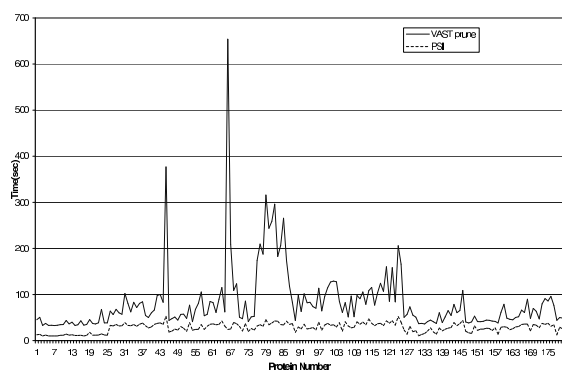
**Figure 10.** The size of the result set obtained by VAST and PSI+VAST for various SCOP classes.

ever, their approach is based on considering  $C_{\alpha}$  atoms, hence much slower than ours. We can do as well by considering SSEs and using smaller index structures.

We also tested PSI to see how it performs as a pruning technique for an existing alignment tool such as VAST. For each query protein in  $D_Q$ , we first ran VAST on  $D_{SF}$ . We stored the results in a set, called  $R_{vast}$ . We also ran PSI for the same query proteins on  $D_{SF}$  to obtain a small candidate set by pruning the proteins in  $D_{SF}$  that has seeds of high p-value. Later, we ran VAST on this candidate set, and stored the results in set  $R_{pruned}$ . We compared  $R_{vast}$  and  $R_{pruned}$  to check whether PSI has dismissed proteins that VAST considers relevant. The results are shown in Figure 10. As we can see from this figure, running the VAST on the whole dataset or on the pruned dataset does not change the result set size significantly. According to this test, PSI has a recall (or sensitivity) of 98.2%. Therefore, we conclude that PSI only eliminates the proteins that are not similar to the query protein.

## 5.2 Performance comparison

We compared the runtime performance of PSI with VAST's pruning step. VAST first finds seeds using SSEs of the query and the target protein. Then it computes p-values corresponding to these seeds. Finally, the promising proteins (based on p-values) are considered for the expensive  $C_{\alpha}$  alignment step. Since PSI aims to optimize the initial pruning, we considered the runtime of only the first two steps of VAST. For all proteins in  $D_Q$ , we ran PSI and VAST on  $D_{SDC}$ . The results for each protein are shown in Figure 11. As can be seen, PSI is faster than VAST for all the proteins. Figure 12 shows a class-wise summary of these results. For all classes, PSI is significantly faster than VAST. We achieved the highest speedup for  $\alpha/\beta$  proteins. This can be explained using Figure 10. The  $\alpha/\beta$  proteins have more neighbors on the average. As a result of this, VAST inspects more seeds in these cases. However, PSI only considers the parts of proteins that are candidates for a



**Figure 11.** Runtime comparisons of VAST prune technique and PSI for each query protein. Target set is single domain protein chains.

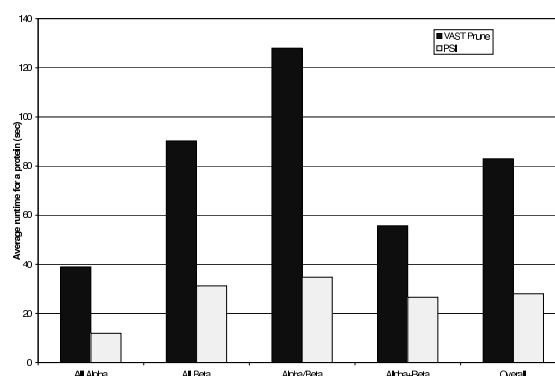
similarity, and finds the seeds in linear time w.r.t. the number of SSEs in the proteins.

Our last experiment considers the scalability of PSI with respect to the dataset size. In this experiment, we created datasets of sizes 2, 4, 8, and 16 times larger than the  $D_{SDC}$  dataset by duplicating  $D_{SDC}$ . This led to a linear increase in the number of relevant proteins as well as the number of irrelevant proteins for each query. Figure 13 compares the average running time of VAST and PSI for the query proteins in  $D_Q$ . The running times increase linearly for our technique as well as for VAST's pruning step. As a result of this, PSI always runs significantly faster than VAST for all dataset sizes. We observed a similar behavior with  $D_{CF}$  dataset.

Our index structure takes only 55 MB of memory and can be constructed in 28.5 minutes for  $D_{SDC}$  dataset. The memory overhead is negligible for current PCs. The time overhead is also insignificant since it is a one time cost and the same index structure would be used for all the queries.

## 6 Joining protein structure datasets

So far, we discussed how to find proteins similar to a single query protein. Here, we extend these techniques to answer join queries on protein structure datasets. Given two protein datasets  $R$  and  $S$ , the join of  $R$  and  $S$  is defined as the set of protein pairs  $(r,s)$ , where  $r \in R$ ,  $s \in S$ , and  $r$  and  $s$  are similar. We represent this set using  $JOIN(R, S)$ . If  $R = S$ , then this is called a *Self Join*. This problem is harder than searching a single query protein since it involves finding similarity to all the proteins in the query dataset. A structure alignment tool such as VAST can answer join queries by comparing every  $r \in R$  with every  $s \in S$ . How-



**Figure 12.** Run time comparisons of VAST prune technique and PSI for various SCOP classes. Target set is the  $D_{SDC}$  dataset.

ever, this will lead to a very slow computation. For example, the pruning step of VAST for self joining  $D_{SDC}$  takes more than two weeks on a 1.6 GHz computer with 1 GB memory. In this section, we will discuss how to accelerate join queries by using the feature vectors of the proteins.

### 6.1 Using feature vectors for join queries

In Section 3, we showed that feature vectors can be used to prune uninteresting regions of the dataset for a given query. We will employ a similar pruning technique to the classic *Nested Loop Join (NLJ)* algorithm [17]. Therefore, we call this technique *PSI-NLJ*.

We start by extracting the feature vectors of every protein in each dataset as explained in Section 2. Later, we create an MBR for the feature vectors of each protein in  $R$  and  $S$ . For a given query  $JOIN(R,S)$ , we fill half of the available memory with the MBRs and the feature vectors of the proteins from  $R$ . Similarly, the other half of the memory is used to store the MBRs and the feature vectors of the proteins in  $S$ . Next, we compare all pairs of MBRs of  $R$  and  $S$  that are in the memory according to three different pruning criteria:

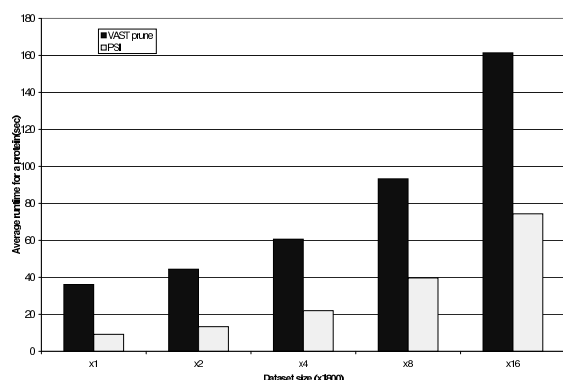
*P1 (Angle test):* Prune if the angles differ by more than  $10^\circ$ .

*P2 (Interval test):* Prune if the min-max intervals differ by more than 20% of the length of the intervals.

*P3 (Length test):* Prune if the ratio of the length of the corresponding SSEs is more than two.

If an MBR pair can not be pruned after the above tests, we perform the same tests on the individual feature vector pairs within these MBRs. In this case, we perform one more pruning test:

*P0 (Type test):* Prune, if the SSEs of a triplet pair are



**Figure 13.** Increase in running time of VAST and PSI with a growth of the protein database.

not of the same type (e.g. prune if one triplet is  $\alpha$ - $\alpha$ - $\alpha$ , and the other triplet is  $\alpha$ - $\beta$ - $\beta$ ). Note that all these pruning steps are also used in single-protein search (see Section 2). For a protein pair, if the number of triplet pairs that remain is larger than some threshold  $T_{min}$  then we use VAST to find the alignment between them. The default value for  $T_{min}$  is 1. As  $T_{min}$  increases, PSI-NLJ eliminates more protein pairs at the cost of being less sensitive. Once we process all the protein pairs in the memory, we read another block of unprocessed MBR set from both datasets and repeat the process.

Constructing the MBRs and extracting the feature vectors take  $O(|R|+|S|)$  time and space, where  $|R|$  and  $|S|$  are the sizes of the datasets. This is a one-time cost. The pruning step takes  $O(|R|\cdot|S|)$  time. However, it is still much faster than VAST since the search is done in feature space. One can also improve the amortized run time complexity of PSI-NLJ to  $O((|R| + |S|)\log(|R| + |S|))$  by building an R\*-tree on the resulting MBRs.

## 6.2 Experimental evaluation

Table 2 shows the number of protein pair comparisons made for self join of  $D_{SDC}$  and  $D_{SF}$  datasets after various pruning steps of PSI-NLJ. As  $T_{min}$  increases, PSI-NLJ prunes more candidate pairs. This is expected since the pruning criteria becomes more stringent. Consecutive rows of this table show the amount of pruning obtained by each pruning test. Let us consider the default case (i.e.  $T_{min} = 1$ ) for  $D_{SDC}$  dataset: type test prunes 10% of the candidates, angle, interval, and length tests prune additional 17.5%, 22.4%, and 28.2% of the candidates. This means that the length test has the highest contribution to the pruning process. The ratio of the initial candidate set size to

**Table 2.** The number of protein pairs that need to be compared after pruning steps P0 to P3 of PSI-NLJ for the self join of the  $D_{SDC}$  dataset. VAST requires 82.1M pairwise comparisons for the same dataset. The numbers in parenthesis show the same value for the self join of the  $D_{SF}$  dataset. VAST requires 1.6M pairwise comparisons for  $D_{SF}$ .

Pruning Test	Number of protein pairs compared		
	$T_{min} = 1$	$T_{min} = 2$	$T_{min} = 3$
P0	73.7M (1.4M)	73.5M (1.4M)	73.5M (1.3M)
P0-P1	59.3M (1.1M)	52.7M (1.0M)	47.6M (0.9M)
P0-P2	40.9M (0.7M)	30.1M (0.5M)	23.5M (0.4M)
P0-P3	17.7M (0.3M)	8.1M (0.1M)	4.4M (0.1K)

pruned candidate set size after all pruning tests is 4.3.

We found the actual run time of VAST's pruning step for self joining of  $D_{SF}$ . It took 14,313 seconds to complete this query. PSI-NLJ computed the same join in only 4,096 seconds where the pruning step of our algorithm constitutes 29 seconds of this time. The remaining 4,067 seconds are spend for VAST's pruning on the remaining set. PSI-NLJ ran 3.5 times faster than VAST. This is consistent with the results we obtained in Table 2.

## 7 Discussion

We considered the problem of similarity searching in protein structure datasets. Our techniques can be used to detect promising proteins for a given query (or a set of queries) quickly.

We proposed to extract feature vectors of the triplets of SSEs. Later, an R\*-tree is built on this feature space. Our first technique, called *PSI*, finds high quality seeds by aligning the SSEs of dataset proteins to a given single query protein. The proteins that do not have high quality seeds are dismissed without further consideration. We also developed a novel statistical model to compute the *p-value* to a seed. This value defines the goodness of this match. Our second technique, called *PSI-NLJ* finds the number of potentially similar triplet pairs by searching the feature space for join queries. Only the protein pairs that have enough similar triplets are used in the actual join operation.

According to our experimental results on the PDB, PSI classified more than 88% of the superfamilies correctly. More than 98% of our results concurred with those of VAST. PSI ran 3 to 3.5 times faster than VAST's pruning step.

Protein structure search is an important emerging application. The explosive increase of the size of the structure databases and the complexity of the search algorithms make faster techniques imperative. The techniques presented in

this paper are an important step in this regard, and will be widely applicable in the field of structural similarity.

## References

- [1] <http://www.rcsb.org/pdb/>.
- [2] <http://alto.rockefeller.edu/modbase-cgi/index.cgi>.
- [3] N.N. Alexandrov and D. Fischer. Analysis of topological and nontopological structural similarities in the PDB: new examples from old structures. *Proteins*, 25:354–365, 1996.
- [4] Mihael Ankerst, Gabi Kastenmüller, Hans-Peter Kriegel, and Thomas Seidl. Nearest neighbor classification in 3D protein databases. In 34–43, 1999.
- [5] K.S. Arun, T.S. Huang, and S.D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, September 1987.
- [6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R\*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, Atlantic City, NJ, 1990.
- [7] O. Çamoğlu, T. Kahveci, and A. Singh. PSI: Indexing protein structures for fast similarity search. In *ISMB*, 2003.
- [8] I. Eidhammer and I. Jonassen. Protein structure comparison and structure patterns – an algorithmic approach. ISMB tutorial, 2001.
- [9] H. Gabow. *Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs*. PhD thesis, Stanford University, 1973.
- [10] M. Gerstein and M. Levitt. Using iterative dynamic programming to obtain pairwise and multiple alignments of protein structures. In *ISMB*, pages 59–66, 1996.
- [11] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233:123–138, 1993.
- [12] L. Holm and C. Sander. 3-D lookup: Fast protein structure database searches at 90 % reliability. In *ISMB*, pages 179–187, 1995.
- [13] I. Koch, T. Lengauer, and E. Wanke. An algorithm for finding maximal common subtopologies in a set of protein structures. *Journal of Computational Biology*, 3(2):289–306, 1996.
- [14] T. Madej, J.-F. Gibrat, and S.H. Bryant. Threading a database of protein cores. *Proteins*, 23:356–369, 1995.
- [15] K. Mizguchi and N. Go. Comparison of spatial arrangements of secondary structural elements in proteins. *Protein Engineering*, 8:353–362, 1995.
- [16] R. Nussinov and H.J. Wolfson. Efficient detection of three-dimensional structural motifs in biological macromolecules by computer vision techniques. *Proc. National Academy of Sciences of the USA*, pages 10495–10499, 1991.
- [17] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. MC Graw Hill, 2000.
- [18] S.D. Ruffino and T.L. Blundell. Structure-based identification and clustering of protein families and superfamilies. *Journal of Computer Aided Molecular Design*, 8:5–27, 1994.
- [19] H.N. Shindyalov and P.E. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Engineering*, 11(9):739–747, 1998.
- [20] A.P. Singh and D.L. Brutlag. Hierarchical protein structure superposition using both secondary structure and atomic representations. In *ISMB*, pages 284–293, 1997.
- [21] W.R. Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Science*, 8:654–665, 1999.
- [22] W.R. Taylor and C.O. Orengo. Protein structure alignment. *Journal of Molecular Biology*, 208:1–22, 1989.
- [23] H.J. Wolfson and I. Rigoutsos. Geometric hashing: An introduction. *IEEE Computational Science & Engineering*, pages 10–21, Oct-Dec 1997.