PSI: Indexing Protein Structures for Fast Similarity Search

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 consider the problem of finding similarities in protein structure databases. Our techniques extract feature vectors on triplets of SSEs (Secondary Structure Elements). Later, these feature vectors are indexed using a multidimensional index structure. Our first technique finds proteins similar to a query protein in a protein dataset. This technique quickly prunes unpromising proteins using the index structure. The remaining proteins are then aligned using a popular 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

The key problem in structural alignment of proteins is to find the optimal correspondence between the atoms in two molecular structures. It is not known which atoms of one structure correspond to the other. 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 correspondence, 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 essentially three classes of algorithms for structural alignment of proteins [7]. The first set performs structural alignment directly at the level of C_α atoms. The second group of algorithms first uses the SSEs (Secondary Structure Elements) to carry out an approximate alignment and then uses the C_α atoms. The final group of algorithms uses geometric hashing [22].

The simplest algorithm for structural alignment [9] uses dynamic programming to find the optimal correspondence. The *DALI* algorithm [10] uses distance matrices to align proteins. The CE algorithm [18] performs a combinatorial extension of aligned fragment pairs. The Double Dynamic Programming algorithm [21] and Iterative Dynamic Programming algorithm [20] use two levels of dynamic programming.

Hierarchical algorithms are based on rapidly identifying correspondences 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. A significant speedup can be obtained since the number of SSEs is small and the 3-D structure within an SSE is constrained by hydrogen bonding. 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, 12, 14, 17, 19].

The *VAST* algorithm [13] 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 protein 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_α 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 align-

ment.

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 [11] defines reference frames using SSEs. Nussinov and Wolfson [15] define reference frames based on C_α atoms. The space complexity of this technique is cubic in the number of elements considered for each target protein.

In this paper, we consider the problem of finding similarities in protein structure datasets. Our techniques can be used to prune uninteresting proteins for a given query (or a set of queries) 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 a given query protein. The proteins that do not have high quality seeds are pruned 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. Protein pairs that do not have enough similar triplets are pruned from the actual join operation.

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.

As the sizes of experimentally determined [1] and theoretically estimated [2] protein structures grow, there is a need for scalable searching techniques. Our algorithm presented here can identify promising matches quickly through the construction of memory-efficient index structures. This tradeoff of query time for a one-time index construction cost is highly desirable in current environments where databases change rather slowly, and queries are frequent. 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

Our construction of index structure is proceeds in four steps: 1) SSE approximation, 2) triplet construction, 3) feature vector extraction, and 4) multidimensional index structure construction

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

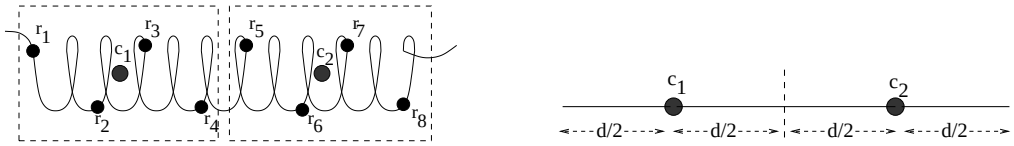
SSE Approximation

Let $a \in \mathcal{D}$ be a protein structure, where $S_a = \{s_1, \cdots, s_{n_a}\}$ is the SSEs of a. Let $R_{s_i} = \{r_{i,1}, r_{i,2}, \cdots, r_{i,\zeta_i}\}$ be the residues that constitute s_i. Here, $r_{i,k}$ corresponds to the k^{th} residue in s_i. We start by splitting R_{s_i} into two equal sized sets as $R^1_{s_i} = \{r_{i,1}, \cdots, r_{i,\zeta_i/2}\}$, and $R^2_{s_i} = \{r_{i,\zeta_i/2+1}, \cdots, r_{i,\zeta_i}\}$. We define c_1 and c_2 as the centers of masses of the residues in $R^1_{s_i}$ and $R^2_{s_i}$. 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 α-helix with eight residues, and Figure 1(b) depicts the line segment approximation to this α-helix. A similar procedure is performed for β-strands.

Triplet construction

For each s_i, we consider the SSEs whose midpoints to the midpoint of s_i are at most 50 $\mathring{A}$. Of these SSEs in the local neighborhood, we consider (at most) four closest ones. The number of nearest neighbors is restricted to four in order to limit the number of triplets and the size of the index structure. Since residues that are close spatially are more valuable for structural similarity [7], the distance to the nearest neighbors is limited to 50 $\mathring{A}$

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



(a) (b)

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

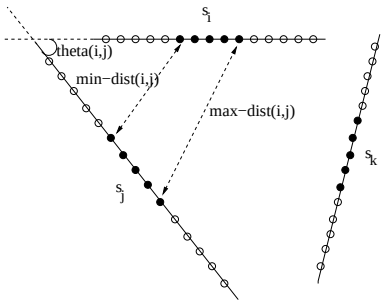


Figure 2. The extraction of feature vector for triplet $<s_i, s_j, s_k>$.

$C_2^{|V|}$, 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 $6n$. Our experiments show that the total number of triplets for the entire PDB [1] is approximately 3.8 times the total number of SSEs.

Feature vector extraction

Let $<s_i, s_j, s_k>$ be a triplet. We start by splitting each SSE in this triplet 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 2 depicts this. The black points are the points selected for each SSE in this triplet. The pair of SSEs, $<s_i, s_j>$, 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) θ_{ij} = the angle between the line segment approximations of s_i and s_j.

Figure 2 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 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.

Index structure construction

For each triplet, we construct an object with 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 double values, the second and third items contain three integer values each, and the fourth item contains five characters. Once the objects for all the feature vectors 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 three 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.

3.1 Finding similar triplets

Let q be the given query protein. We start by extracting all the feature vectors of q 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 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·($max_{q_i,q_j} - min_{q_i,q_j}$); for all i, j;
2. $\Delta(\theta)$ = 10°;
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 θ_{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 ($0.5 < \frac{length(p_i)}{length(q_i)} < 2$) for all i then
           – $S(t_q, t_d)$:= *TRIPLET-PAIR-SCORE(*t_q, t_d*)*;

Figure 3. Range query algorithm.

Figure 3 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 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 speed of the algorithm. They are set to 0.5 and 2 based experimental results. The similarity score for triplet pairs is based on RMSD distance between the midpoints of SSEs of corresponding triplets.

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. Longer mappings of SSEs can be found by merging the results of all the pairs. We capture the correlation between SSE pairs by building a *Triplet Pair Graph (TPG)* on similar triplet pairs. The vertices of 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* Product 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 w_i = 1- (number of triplets in the dataset that align to the query triplet in v_i)/(total number of triplets in the dataset).

We run *Depth First Search (DFS)* algorithm on the TPG to find the *Largest Weight Connected Component (LWCC)*. 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 SSEs of target proteins and the query 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 contain the SSE pair $<q_i p_j>$ in G'.

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 [8] 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.

4 Evaluating seeds

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.

Each seed is represented with seven numbers: the number of α-helices and β-strands in query and target proteins (4), the number of SSEs aligned of each type (2), 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 α-helices, and b_1 and b_2 β-strands respectively. Let ψ be the seed with a α matches and b β matches having an RMSD of d, then the p-value of ψ is defined as*

p-value(ψ) = *The probability that a random alignment of these two proteins contains at least a α-helices and at least b β-strands within distance d.*

Consider an example seed with five α-helices in query protein and four α-helices in target protein. For simplicity, each SSE is represented by a point in 3-dimensional Euclidean space. Assume that three α-helices are aligned with RMSD = d. We assume that the distance between any two aligned SSEs is not greater than the RMSD of the seed. Figure 4 illustrates this example in 2-D. Here, black circles represent query SSEs, and black rectangles represent target SSEs. The circles around 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 around that query SSE. Only three SSEs are aligned in Figure 4.

Given two input proteins, we build a binary probability tree. Figure 5 depicts a probability tree for a_1 = 5, and a_2 = 4, where a_1 and a_2 represent the number of α-helices in the proteins. The nodes at the k^{th} level show the contribution of the k^{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 simply 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

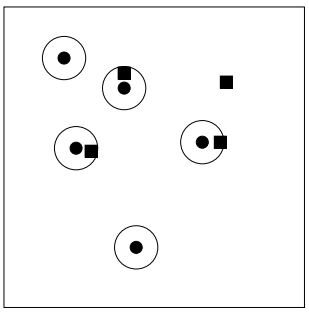


Figure 4. Illustration of a sample seed in 2-D. The big box represents the whole search space. Black circles represent query SSEs. Black rectangles represent target SSEs. Here, only three SSEs are aligned. The circles around query SSEs have a radius equal to the RMSD of this alignment.

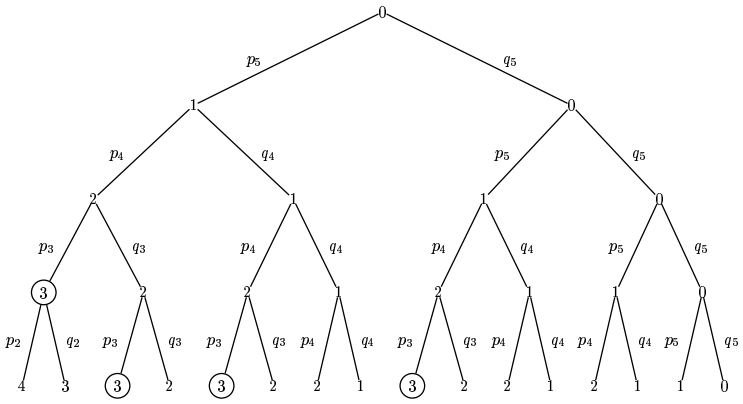


Figure 5. The probability tree for the alignment of a query protein with five α-helices and a target protein with four α-helices. The probabilities of the circled nodes are accumulated to find the probability of aligning three α-helices.

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 probabilities of the nodes with value k, and which do not have an ancestor of value k. For example, in Figure 5, the probability of aligning at least three α-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 simple dynamic programming strategy.

The p-value of a seed can be calculated as the multiplication of the probabilities of the alignment of α-helices and β-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 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 α, all β, α+β and α/β. 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 has been shown in Table 1. The tests are run on an 1.6 GHz AMD computer with 1GB memory.

5.1 Quality comparison

Our first experiment set inspects the quality of the seeds found using the feature vectors. We classify the query protein into one of the superfamilies using the k best results in feature space 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 6 shows the percentage of query proteins correctly classified for different number of nearest neighbors, using S_{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 slightly for 3-NN, but the percentage drops for larger number of results. Even for 20-NN, more than 76% of the proteins are classified correctly.

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	7i1b	1ba7-B	1inc	3hvp	1i7a-A	1ief-B	1acd
61-70	2iza	1dyw-A	1bnu	1bwu-P	1hg8-A	1thj-C	1cax-F	1hjg-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	1tyf-L	1c2y-P
91-100	1f1j-B	2chf	1fln	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	4rnt	205l	1au0	1bxi-B	1rbg	1e1s-A
131-140	1ejr-A	1qg7-B	1azq-A	3rhn	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	1otf-D	1icr-B	4aig	1bkl
161-170	2hpr	1b9l-A	1i1d-A	1hqz-8	1fil	1byw-A	1ga7-A	1b5m	1i5c-B	1qmr-A
171-179	1f7l-A	1g3i-R	1aha	1prt-G	1bnl-A	1fzd-B	1gu9-C	1jya-A	1is8-K	1lep-E

Table 1. The PDB ids of queries used in our experiments.

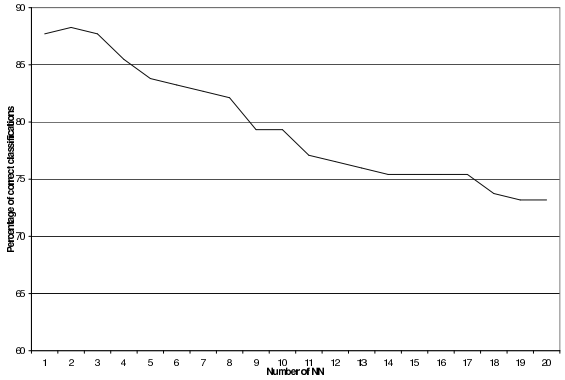


Figure 6. The percentage of proteins correctly classified using seed for different number of NN.

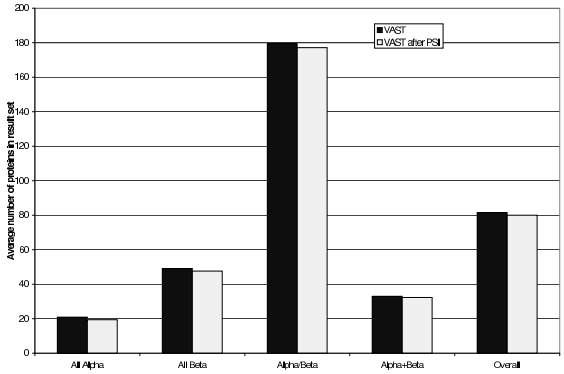


Figure 7. The size of the result set obtained by VAST and PSI+VAST for various SCOP classes.

Ankerst et al [4] built an index structure on 3-D shape histograms and considered the similar problem of protein classification. Their accuracy is similar to ours. However, their approach is based on considering C_α atoms. 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 protein in D_Q, we first ran VAST on D_{SF}, and saw how many proteins are returned in the answer set. We also ran PSI for the same protein on D_{SF} to obtain a candidate set. Later, we ran VAST on this set. We compared these two results to check whether PSI has pruned proteins that VAST considers relevant. The results are shown on Figure 7. 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%.

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 protein. Then it computes p-values corresponding to these seeds. Finally the promising proteins (based on p-values) are considered for the expensive C_α 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 8. As can be seen, PSI is faster then VAST for all the proteins. Figure 9 shows a class-wise summary of these results. For all classes, PSI is significantly faster than VAST. But speedup is the

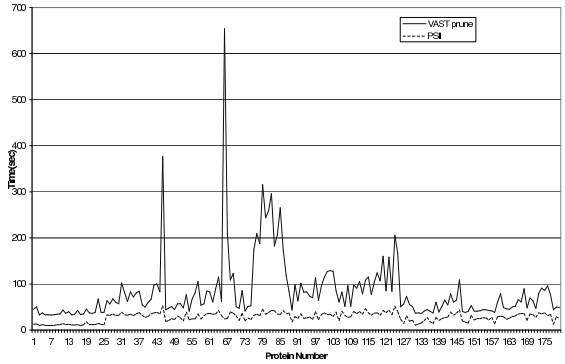


Figure 8. Runtime comparisons of VAST prune technique and PSI for each query protein. Target set is single domain protein chains.

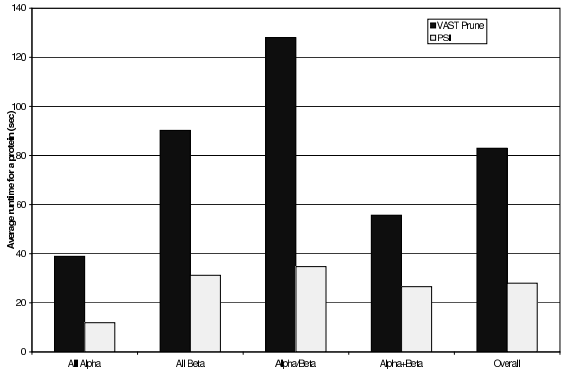


Figure 9. Run time comparisons of VAST prune technique and PSI for various SCOP classes. Target set is the D_{SDC} dataset.

greatest for α/β proteins. This can be explained using Figure 7. The α/β proteins have more neighbors on the average. Because of that, VAST needs to inspect more seeds in these cases. However, PSI only considers the parts of proteins that are candidates for a similarity, and finds the seeds in linear time w.r.t. the number of SSEs in the proteins.

Our last experiment considers the effect of increasing the database size on the running time of the programs. For this, we added several copies of the same dataset, D_{SDC}, as indicated in Figure 10. This led to a linear increase in the number of relevant proteins as well as the number of irrelevant proteins for each query. As expected, the running times increase linearly for our technique as well as for VAST's pruning step. We observed a similar behavior with D_{CF} dataset.

Our index structure takes 55 MB of memory and can be constructed in 28.5 minutes for D_{SDC} dataset. The memory overhead is negligible for current PCs and the time overhead is a one time cost.

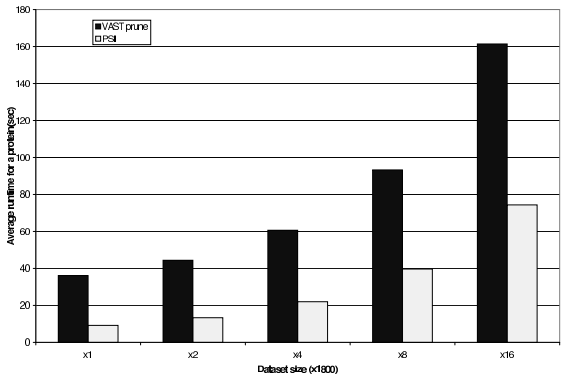


Figure 10. Increase in running time of the techniques with a growth of the protein database.

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 with every s. But 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 1.4 GHz computer with 1GB 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 [16]. 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. 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°.

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 not of the same type (e.g. prune if one triplet is α-α-α, and the other triplet is α-β-β.). 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. 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 by building an R*-tree on the resulting MBRs.

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 (68.1K)

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}.

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 pruned candidate set size after all pruning tests is approximately 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. This is consistent with the results we obtained in Table 2.

7 Discussion

In this paper, we considered the problem of similarity searching in protein structure datasets. Our

techniques can be used to prune unpromising 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 pruned 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. Protein pairs that do not have enough similar triplets are pruned from 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 makes faster techniques imperative. The techniques presented in this paper are an important step in this regard, and will be widely applicable in structural similarity field.

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] I. Eidhammer and I. Jonassen. Protein structure comparison and structure patterns – an algorithmic approach. ISMB tutorial, 2001.

[8] H. Gabow. *Implementation of Algotihms for Maximum Matching on Nonbipartite Graphs*. PhD thesis, Stanford University, 1973.

[9] M. Gerstein and M. Levitt. Using iterative dynamic programming to obtain pairwise and multiple alignments of protein structures. In *ISMB*, pages 59–66, 1996.

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

[11] L. Holm and C. Sander. 3-D lookup: Fast protein structure database searches at 90 % reliability. In *ISMB*, pages 179–187, 1995.

[12] 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.

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

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

[15] R. Nussinov and H.J. Wolfson. Efficient detection of threedimensional structural motifs in biological macromolecules by computer vision techniques. *Proc. National Academy of Sciences of the USA*, pages 10495–10499, 1991.

[16] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. MC Graw Hill, 2000.

[17] S.D. Rufino and T.L. Blundell. Structure-based identification and clustering of protein families and superfamilies. *Journal of Computer Aided Molecular Design*, 8:5–27, 1994.

[18] 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.

[19] A.P. Singh and D.L. Brutlag. Hierarchical protein structure superposition using both secondary structure and atomic representations. In *ISMB*, pages 284–293, 1997.

[20] W.R. Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Science*, 8:654–665, 1999.

[21] W.R. Taylor and C.O. Orengo. Protein structure alignment. *Journal of Molecular Biology*, 208:1–22, 1989.

[22] H.J. Wolfson and I. Rigoutsos. Geometric hashing: An introduction. *IEEE Computational Science & Engineering*, pages 10–21, Oct-Dec 1997.