JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 12, Number 9, 2005

Pp. 1221–1241

Automatic 3D Protein Structure Classification without Structural Alignment

ZEYAR AUNG and KIAN-LEE TAN

ABSTRACT

In this paper, we present a new scheme named ProtClass for automatic classification of three-dimensional (3D) protein structures. It is a dedicated and unified multiclass classification scheme. Neither detailed structural alignment nor multiple binary classifications are required in this scheme. We adopt a nearest neighbor-based classification strategy. We use a filter-and-refine scheme. In the first step, we filter out the improbable answers using the precalculated parameters from the training data. In the second, we perform a relatively more detailed nearest neighbor search on the remaining answers. We use very concise and effective encoding schemes of the 3D protein structures in both steps. We compare our proposed method against two other dedicated protein structure classification schemes, namely *SGM* and *CPMine*. The experimental results show that ProtClass is slightly better in accuracy than SGM and much faster. In comparison with CPMine, ProtClass is much more accurate, while their running times are about the same. We also compare ProtClass against a structural alignment-based classification scheme named *DALI*, which is found to be more accurate, but extremely slow. The software is available upon request from the authors. The supplementary information on ProtClass method can be found at: *http://xena1.ddns.comp.nus.edu.sg/∼genesis/PClass.htm*.

Key words: protein structure, abstract representation, filter-and-refine, nearest neighbor classification.

1. INTRODUCTION

ANALYZING THREE DIMENSIONAL (3D) PROTEIN STRUCTURES is an important task in bioinformatics. Analysis of protein structures can give insights into the functions of proteins, which are useful in many end-user applications such as drug discovery. Protein structure analysis involves such tasks as protein structure comparison, classification, homology modeling, and prediction (Orengo *et al.*, 2003), and researchers have been developing numerous automated methods for structural comparison, homology modeling, and prediction in the last decade. But only a few automated structural classification methods have been proposed up until now. Instead, people have relied on the manual and semi-automatic classification methods such as *SCOP* (Hubbard *et al.*, 1997) and *CATH* (Orengo *et al.*, 1997). Or they use the traditional structural

Department of Computer Science, National University of Singapore, Singapore 117543.

comparison or alignment methods such as *DALI* (Holm and Sander, 1993), *CE* (Shindyalov and Bourne, 1998), *VAST* (Gibrat *et al.*, 1996), and *CMO* (Caprara *et al.*, 2004) for classification purpose.

Due to advancements in laboratory methods to determine the 3D structures of proteins, protein structure databases such as *PDB* (Berman *et al.*, 2000) are growing rapidly in size. For example, the total number of protein structures in PDB was about 4,000 ten years ago (in 1995), but it is about 32,000 now (July 2005). Thus, determining the respective structural classes of these newly added protein structures by the manual or the semi-automatic methods becomes inadequate. (Here, the term "class" in the computer science literature refers to a group with common physical or functional properties in general and should not be confused with the term "Class" in SCOP hierarchy.) Classifying a new protein structure through traditional structural alignment/comparison also becomes inefficient because it involves the comparison of the new protein against all the proteins (or a representative set) in the large database. Although faster database searching methods such as *PSI* (Camoglu *et al.*, 2003) and *ProtDex2* (Aung and Tan, 2004b) are available, they are not designed specifically for classification, and thus still require quite some time to search through the entire database.

Our objective is to develop an efficient, dedicated and automated protein structure classifier without having to perform any detailed structural alignment/comparison or detailed database searching. We propose a new classification scheme named ProtClass, which simply stands for protein classification.

Suppose we have a database of known protein structures. We build our classifier by using these structures in the database as the training data. Here, the class labels of the training protein structures must be known in advance. We use SCOP class labels in our scheme. (SCOP is a manually constructed protein structure classification system in which the human experts assign the hierarchical class labels—Class, Fold, Super Family and Family—to the known protein structure domains in the PDB database. It is regarded as a golden standard by biologists. But, being manual, its classification process is slow and tedious.) We can alternatively use CATH class labels and domain definitions by tuning a few parameters in our scheme. (CATH, another widely used protein structure classification system, is semi-automatically constructed.)

Then, we train our classifier by encoding the protein structures in each class into their concise formats and extracting some important pieces of information from each distinct class. In this way, we can explore prior knowledge and expert human judgement in classifying proteins and exploit this knowledge for automatically classifying new protein structures in the future. (Traditional alignment-based methods do not exploit such expert knowledge when used as classification tools.) When we want to classify a new protein structure whose class is unknown, we also represent it in its concise format, perform filtration using the acquired parameters from the training data, and employ a nearest neighbor search in order to efficiently and effectively predict the possible class label(s) for it.

We conduct an experiment performing 10-fold cross validation on a dataset of 600 proteins. The results show that we can classify an unknown protein structure with very good accuracy within a very short time. We compare ProtClass against two other dedicated protein structure classification schemes, namely, *SGM* (Røgen and Fain, 2003) and *CPMine* (Aung and Tan, 2004a). The experimental results show that ProtClass is slightly better in accuracy than SGM and much faster. In comparison with CPMine, ProtClass is much more accurate, while their running times are about the same. We also compare ProtClass against a DALI structural alignment-based classification scheme, which is found to be more accurate, but extremely slow.

The good performance of ProtClass is attributed to its concise and effective protein structure encoding scheme and its simple yet powerful nearest neighbor search algorithm. The ideas of concise encoding and nearest neighbor search are inspired by those of SGM.

The rest of the paper is organized as follows. Section 2 gives some background information relevant to the proposed scheme. Section 3 describes how 3D protein structures can be represented in their concise formats. Section 4 describes how the classifier is built from the training data and how new unknown proteins are classified. Section 5 presents the experimental results and discussions, and Section 6 concludes the paper.

2. BACKGROUND

This section discusses the basic concepts regarding classification of protein structures and some of the previous works that are related to the proposed scheme.

2.1. Protein structure classification

According to the computer science literature, classification, in general, is a kind of supervised learning. First, we have a collection of objects with their class labels already known. Then, for each distinct class, we train the classifier with positive examples of objects that belong to the class and, optionally, negative examples of objects that do not belong to the class. The classifier learns the classification rules from the training data and reapplies these rules to classify the new unknown objects. *Nearest neighbor search (NN)*, *support vector machines (SVMs)*, *neural networks*, and *Bayesian networks* are some commonly used classification methods.

In our case, we have a database of 3D protein structures with their structural class labels already known (according to SCOP classification). For each unique structural class, we train our classifier with positive and, optionally, negative examples of structures. We predict the structural classes of newly determined structures using the rules learned by the classifier from the training examples.

Structural classification is different from structural clustering schemes such as Fischer *et al.* (1995) in which unsupervised learning is used to build clusters of protein structures whose class labels are not known in advance.

2.2. Structural classification versus structural comparison

Structural classification is also a different problem from structural alignment/comparison and structural database searching. First, in classification, the class labels of the proteins in the database (training data) must be known in advance. Thus, we can utilize knowledge about the characteristics of the classes in the training data in various ways when classifying new proteins whose class labels are unknown yet. Although we may know the class labels of the database proteins in advance in structural alignment or database searching, this information is not utilized.

In our proposed ProtClass scheme, the system learns and capitalizes the distance threshold parameters from each of the classes. So, it can be categorized as a supervised classification scheme, rather than a mere database searching scheme which does not use any information of the classes' characteristics.

Second, the objective of classification is only to predict the possible class(es) of an unknown protein. In contrast, that of structural alignment/comparison is to align or compare two given proteins according to a similarity metric such as RMSD, and that of database searching is to search all or a majority of similar proteins with respect to a given query. The similarity metric used by a structural alignment or database searching method must be able to capture the similarity between the query and the distantly related protein structures so that they can also be included in the answer along with the closely related ones. Although a database search may be involved in a classification process (as in a nearest neighbor classification scheme), only some adjacent neighbors to the query are used to predict the possible class(es) of it, and the distantly related proteins are simply ignored, even though they may belong to the right class. Thus, the data representation and the similarity metric for structural classification can be coarse as long as they can capture the similarity of the query protein structure and a few of its nearest neighbors. Given sufficient and representative training data for each class, it is almost always possible to find the correct class of the query by looking at a few nearest neighbors. (A classification system, irrespective of the method used, will not perform well if the training data is insufficient, biased, or unrepresentative.)

Although pairwise structural alignment methods such as DALI and CE can be used as the basic nearest neighbor search tools against the whole database, it is a sort of overkill because many of the detailed alignment results (with proteins unrelated or distantly related to the query) are totally irrelevant to the classification result. As such, these methods are prohibitingly expensive to run.

For example, it is observed in our experiments that our ProtClass classifier takes only 4 minutes and 14 seconds to run a classification cycle on 540 training proteins and 60 query proteins on an average standalone machine. But, running the same task with DALI on the same machine is about 640 times slower, and takes about 1 day and 21 hours (see Section 5). This running time seems still affordable. However, we should note that the above-mentioned classification task involving some hundreds of proteins is only a medium one. It can be speculated that DALI will take an unbearably long time for a large classification task involving tens of thousands of proteins.

2.3. Related work

There are quite a number of protein structure classification methods such as those of Ding and Dubchak (2001) and Chinnasamy *et al.* (2004) that use the amino acid (AA) sequences as their input data. But, on the other hand, to our knowledge there exist only a few pure protein structure classifiers that take the 3D structures as their input data. As mentioned above, many people use the traditional structural alignment methods such as DALI (Holm and Sander, 1993) and CE (Shindyalov and Bourne, 1998) for classification purposes. (Interested readers can refer to the survey papers by Lancia and Istrail [2004] and Eidhammer *et al.* [2000] for details on structural alignment/comparison methods.) The alignment-based classification approach incurs high computational cost unnecessarily because structural alignment is a computationally expensive task. A pure or dedicated classifier does not need to carry out any detailed alignment/comparison and can accomplish the required classification task with much lower computational cost. We show the comparison of DALI against our proposed ProtClass scheme in Section 5.

Huan *et al.* (2004) recently proposed a dedicated protein structure classifier based on *coherent subgraph analysis*. It first presents a 3D protein structure as a feature vector of the number of occurrences of the coherent subgraphs representing the peptide and proximity relations among the amino acid residues and then uses SVMs to train and classify these feature vectors. The software for the system is not readily available for investigation, and we cannot directly compare it against our scheme.

SGM (Røgen and Fain, 2003) is another recent dedicated protein structure classifier that uses *Gauss integrals* to represent a 3D protein structure as a 30-dimensional feature vector. For each structural class, it constructs a cluster containing the member proteins in 30-dimensional space. It then performs a nearest cluster search to predict the most appropriate class for a new unknown protein. Its source code is available, and we compare it against our proposed scheme in Section 5.

We also recently proposed a dedicated protein structure classification scheme named *CPMine* (Aung and Tan, 2004a), which presents 3D protein structures as CPsets (sets of feature vectors representing inter-SSE contact patterns) and mines the sub-CPsets that occur frequently in their respective classes. These frequent sub-CPsets are regarded as the classes' fingerprints which can be used to indicate the possible class(es) of an unknown protein. We also compare it against our proposed scheme in Section 5.

3. ENCODING PROTEIN STRUCTURES

In this section, we will discuss how 3D protein structures can be concisely and effectively represented in their encoded formats, namely *PA* and *CPset*, and how the distances between these encoded formats are measured.

3.1. Protein abstract (PA)

Let P be a 3D protein structure. Let $A = a_1a_2a_3 \ldots a_{|A|}$ be the amino acid (AA) residue sequence of P where $|A|$ is the number of residues in P. Let $S = s_1s_2s_3 \ldots s_{|S|}$ be the SSE sequence of P where $|S|$ is the number of SSEs in P. (A secondary structure element (SSE) is a conserved structural pattern within the overall 3D structure of a protein. Helix (H) and sheet (E) are two common types of SSEs. We use the *Stride* algorithm [Frishman and Argos, 1995] to extract the SSE information from a 3D protein structure file in PDB format.) As an example, let us assume we have a 10-residue protein P with the following AA sequence A.

K F A V N H I T R S

Let us also assume that we have three SSEs in the protein where residues 2–3 forms a sheet, 5–6 a helix, and 8–10 a sheet. Then we have the SSE sequence S of P as follows.

E H E

Here we formulate a concise encoding format named *protein abstract* (PA). It is a simple tuple featuring six attributes regarding the overall structure of a 3D protein structure as shown in Table 1.

TABLE 1. ATTRIBUTES IN A PROTEIN ABSTRACT (PA)

Sr	*Description*	*Symbol*	*Equation*	*Example for protein P*
1	No. of AA residues	$\|A\|$		10
2	No. of SSEs	$\|S\|$		3
3	Total length of all SSEs as a percentage of no. of residues	*SL*	(1)	0.7
4	Total length of all helices as a percentage of total SSE length	*HL*	(2)	0.29
5	No. of helices as a percentage of no. of SSEs	*HN*	(3)	0.33
6	SSE sequence	*S*		E H E

Attributes 1, 2, and 6 are readily available from the PDB file and the Stride output. Attributes 3–5 can be calculated using the following equations:

$$SL = \left(\sum_{i=1}^{|S|} |s_i| \right) / |A| \tag{1}$$

where $|s_i|$ is the length of SSE s_i. Here, the length of an SSE is an integer in terms of the number of AA residues it constitutes (rather than its physical length in Angstroms (Å)).

$$HL = \left(\sum_{i=1}^{|S|} t_i \right) / SL \quad \text{where} \quad t_i = \begin{cases} |s_i| & s_i \text{ is H (helix)} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$HN = \left(\sum_{i=1}^{|S|} t_i \right) / |S| \quad \text{where} \quad t_i = \begin{cases} 1 & s_i \text{ is H} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

All the above-mentioned six attributes in a PA are designed to capture the overall sequence and structural information of a protein. They can be used to roughly distinguish a protein of one class from one of another. The differences of the attribute values in the PAs of two proteins from the same class tend to be lower than those from different classes. For example, the proteins belonging to the All-α Class usually have very high *HL* and *HN* values as opposed to the All-β Class proteins which usually have very low *HL* and *HN* values. In a big enough class, there almost always exists another protein whose PA attributes are very similar to those of a protein in question. We use this property of the protein structure classes to filter out the proteins which have very different PA attribute values from the query protein.

We can formally define a PA as the following hexa-tuple.

$$\mathcal{PA} = \{|A|, |S|, SL, HL, HN, S\}$$

Given a query protein Q and a database protein P, we can represent them as two PAs: $\mathcal{PA}_Q$ and $\mathcal{PA}_P$. Let b be any attribute in $\mathcal{PA}$. The normalized distance or difference Δ_b between two attribute values (for the first five attributes) belonging to $\mathcal{PA}_Q$, and $\mathcal{PA}_P$, respectively, can be defined as follows.

$$\Delta_b(\mathcal{PA}_Q, \mathcal{PA}_P) = \frac{|b_Q - b_P|}{b_Q} \quad \text{where} \quad b \in \{|A|, |S|, SL, HL, HN\} \tag{4}$$

For the last SSE sequence attribute S, we use ΔS as its SSE *edit distance* which is defined as follows:

$$\Delta_S(\mathcal{PA}_Q, \mathcal{PA}_P) = 1.0 - \frac{\mathbf{NW}(S_Q, S_P)}{|S_Q|} \tag{5}$$

where **NW** is the pairwise SSE alignment score for S_Q and S_P using the Needleman–Wunsch algorithm (a text book algorithm which can be found in Gusfield [1997]).

Suppose we have the precalculated difference (or distance) threshold values δ_b for all six PA attributes. (The threshold value for each attribute differs from class to class. We will discuss how to calculate these threshold values for each of the distinct classes from the training data in Subsection 4.1.) When comparing a pair of PAs (one from Q and one from P), they are deemed similar if and only if $\Delta_b \leq \delta_b$ for every b. The similarity between two PAs can be formally defined as

$$Similar(\mathcal{PA}_Q, \mathcal{PA}_P, \delta) = \begin{cases} \text{TRUE} & \text{if} \quad \Delta_b(\mathcal{PA}_Q, \mathcal{PA}_P) \leq \delta_b \quad \forall b \in \{|A|, |S|, SL, HL, HN, S\} \\ \text{FALSE} & \text{otherwise} \end{cases} \tag{6}$$

where δ is the set of PA attributes' distance thresholds for the class to which the database protein P belongs.

After we have determined that a pair of PAs is similar, we can calculate the *overall distance PADist* between them as a simple Euclidean distance as follows. (We do not have to calculate *PADist* for the two PAs that are not similar. On average, for a PA of a query protein, about 71% of the PAs in the database are dissimilar and can be discarded right away.)

$$PADist(\mathcal{PA}_Q, \mathcal{PA}_P, \delta) = \left(\sum_{b \in \{|A|, |S|, SL, HL, HN, S\}} \left(\frac{\Delta_b(\mathcal{PA}_Q, \mathcal{PA}_P)}{\delta_b} \right)^2 \right)^{0.5} \tag{7}$$

Here, dividing the actual distance Δ_b with distance threshold δ_b maps the normalized distance into the range of 0 to 1 for all attributes. We assume equal weights for all the attributes.

3.2. Discrete contact pattern feature vector set (CPset)

In this subsection, we will discuss how a 3D protein structure can be represented as another abstract structure called CPset, which is a set of integer-valued contact pattern feature vectors.

3.2.1. Distance matrix. A 3D protein structure can be represented as a 2D *distance matrix*. A distance matrix D for a protein P with $|A|$ residues is an $|A| \times |A|$ matrix storing the interatomic distances between the C_α (central carbon) atoms of the AA residues in the protein. A distance matrix is rotation and translation invariant. For example, for our sample protein P from the above section, we can construct its distance matrix D like the one shown in Fig. 1(a). Each cell $D[i, j]$ represents the C_α–C_α distance in Angstroms (Å) between residue i and residue j ($i, j \leq |A|$).

3.2.2. Contact pattern (CP). Generally, the forms and arrangement (topology) of the SSEs are mainly responsible for the overall 3D shape of a protein. When a 3D protein structure is presented as a 2D distance matrix, the forms and arrangement of the SSEs are captured in the submatrices called *contact patterns* (CPs). A CP is formed by the interaction of two SSEs. If we have $|S|$ SSEs in a protein, there will be $|S|^2$ CPs. The CPs for our example protein P are shown as the light and dark gray blocks in Fig. 1(b).

Among the CPs in a distance matrix, only those above the main diagonal are sufficient to capture the forms and arrangement information of the SSEs. This is because the CPs below the main diagonal are just the mirror images of those above the main diagonal, and those on the main diagonal are merely the self-interactions of the single SSEs. In our example, only the CPs C^{12}, C^{13}, and C^{23} (shown as the light gray blocks in Fig. 1(b)) need to be taken into account. The number of such CPs is $|S|(|S| - 1)/2$.

3.2.3. CP feature vector. Now, we extract the important properties from CPs and represent them as a *feature vector*. (This representation scheme is an enhancement of the one used in our previous works (Aung and Tan, 2004a, 2004b). The expression power of the feature vector has been much improved in the current scheme.) There are 10 attributes (properties) in the feature vector we use to represent a CP. These feature vector attributes are designed to effectively determine the similarity or dissimilarity of the CPs they represent. For example, the two CPs with very different mean C_α–C_α distance values cannot be similar in any way. CP feature vector representation is rotation and translation invariant, as it is based on a distance matrix.

(a)

	1	2	3	4	5	6	7	8	9	10
1	0.0	3.8	6.3	8.7	11.9	15.4	18.5	21.3	24.9	27.9
2	3.8	0.0	3.7	6.5	8.9	12.5	15.4	18.2	21.7	24.6
3	6.3	3.7	0.0	3.8	6.2	9.9	13.1	15.4	19.2	21.9
4	8.7	6.5	3.8	0.0	3.8	6.8	10.3	12.9	16.6	19.7
5	11.9	8.9	6.2	3.8	0.0	3.8	6.9	9.4	13.1	16.1
6	15.4	12.5	9.9	6.8	3.8	0.0	3.9	6.3	9.9	13.2
7	18.5	15.4	13.1	10.3	6.9	3.9	0.0	3.8	6.7	10.2
8	21.3	18.2	15.4	12.9	9.4	6.3	3.8	0.0	3.8	6.9
9	24.9	21.7	19.2	16.6	13.1	9.9	6.7	3.8	0.0	3.8
10	27.9	24.6	21.9	19.7	16.1	13.2	10.2	6.9	3.8	0.0

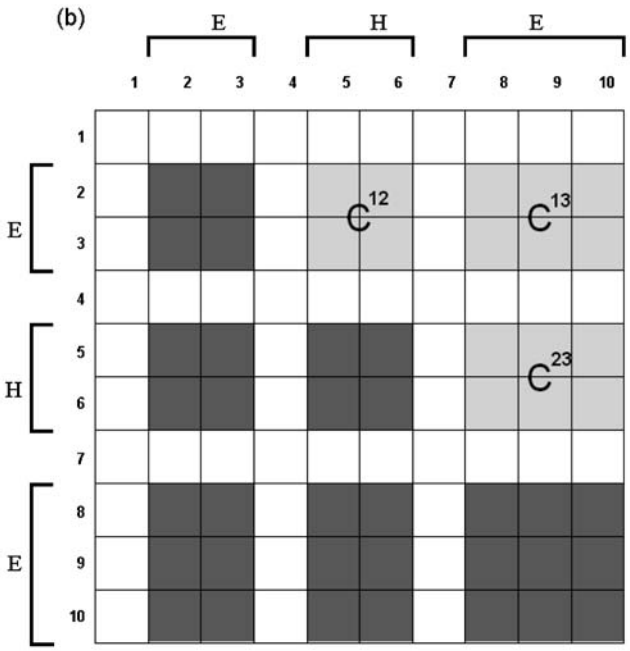


FIG. 1. Example of (**a**) distance matrix and (**b**) contact patterns.

The feature vector of a CP C^{ab}, which is formed by the interaction of s_a (first SSE) and s_b (second SSE), where $b > a$, consists of 10 attributes as shown in Table 2. It is observed that all these 10 attributes are important on their own, because dropping any of them more or less degrades the accuracy of the system as shown in Subsection 5.6.

Attributes 1–5 simply correspond to the types and the positions of the SSEs that make up the CP. The attributes *AS* (AA sequence position of the first SSE) and *SS* (SSE sequence position of the first SSE) can be easily extracted from the PDB file and the Stride output, respectively. The attributes *CT* (contact pattern type), *AD* (AA sequence position difference of two SSEs), and *SD* (SSE sequence position

TABLE 2. ATTRIBUTES OF CP FEATURE VECTOR

Dimension	*Attribute*	*Symbol*	*Equation*	*Upper bound*	*Discretization Parameters*	
					# bins	*# bits*
1	Type of C^{ab}	*CT*	(8)	3	4	2
2	Starting position of s_a in AA sequence A	*AS*	s_a^{start}	800	2	1
3	Position of s_a in SSE sequence S	*SS*	a	48	12	4
4	Difference between starting positions of s_a and s_b in AA sequence A	*AD*	(9)	800	4	2
5	Difference between positions of s_a and s_b in SSE sequence S	*SD*	(10)	48	20	5
6	Torsion angle between $\mathbf{V}_a$ and $\mathbf{V}_b$ (-180.0 to $+180.0$)	Ω	Sfyrakis (2004)	360.0	16	4
7	Closest segment–segment distance of $\mathbf{V}_a$ and $\mathbf{V}_b$	*ND*	Sunday (2004)	64.0	16	4
8	Nearest vertex pair distance $\mathbf{V}_a$ and $\mathbf{V}_b$	*VD*	(11)	64.0	4	2
9	Mean of C_α–C_α distances in C^{ab}	*MD*	(12)	64.0	4	2
10	Contact density of C^{ab}	*CD*	(13)	1.0	2	1
	Total					27

difference of two SSEs) can be simply calculated, respectively, as follows.

$$CT(s_a, s_b) = \begin{cases} 0 & \text{if } (s_a \text{ is H}) \wedge (s_b \text{ is H}) \\ 1 & \text{if } (s_a \text{ is H}) \wedge (s_b \text{ is E}) \\ 2 & \text{if } (s_a \text{ is E}) \wedge (s_b \text{ is H}) \\ 3 & \text{if } (s_a \text{ is E}) \wedge (s_b \text{ is E}) \end{cases} \tag{8}$$

$$AD(s_a, s_b) = s_b^{start} - s_a^{start} \tag{9}$$

where s_a^{start} and s_b^{start} are the starting positions of s_a and s_b, respectively, in AA sequence A. Since $b > a$, $s_b^{start} > s_a^{start}$,

$$SD(s_a, s_b) = b - a \tag{10}$$

where a and b are the positions of s_a and s_b, respectively, in SSE sequence S.

Attributes 6–8 correspond to the *3D vector representation* of the SSEs. An SSE can be roughly approximated by its representative vector or line segment in 3D space. Since a CP represents the interaction between the two SSEs, we can logically associate it with the spatial relationship (torsion angle and distances) between the two SSE vectors.

Let $\mathbf{V}_a$ denote the 3D vector of s_a, and $\mathbf{V}_b$ that of s_b. We can calculate the vertex points ($\mathbf{V}_a^{start}, \mathbf{V}_a^{end}$) and ($\mathbf{V}_b^{start}, \mathbf{V}_b^{end}$) of the vectors using the equations given by Singh and Brutlag (1997).

Now, we calculate the torsion angle attribute (Ω) using the formula given in Sfyrakis (2004) and the closest segment-to-segment distance attribute (*ND*) using the algorithm described in Sunday (2004). The nearest vertex pair distance (*VD*) can be calculated as follows:

$$VD(\mathbf{V}_a, \mathbf{V}_b) = \min \begin{cases} Dist(\mathbf{V}_a^{end}, \mathbf{V}_b^{start}) \\ Dist(\mathbf{V}_a^{start}, \mathbf{V}_b^{end}) \\ Dist(\mathbf{V}_a^{end}, \mathbf{V}_b^{end}) \\ Dist(\mathbf{V}_a^{start}, \mathbf{V}_b^{start}) \end{cases} \tag{11}$$

where *Dist* is the Euclidean distance between two 3D points.

The rest of the attributes are derived directly from the distance matrix D. They are related to the C_α–C_α interaction patterns within a CP. The functions to calculate these attribute values—*MD* (mean of C_α–C_α distances) and *CD* (contact density)—are defined as follows:

$$MD(s_a, s_b) = \frac{\sum_{i=0}^{|s_a|-1} \sum_{j=0}^{|s_b|-1} D[s_a^{start} + i, s_b^{start} + j]}{|s_a||s_b|}, \tag{12}$$

$$CD(s_a, s_b) = \frac{\sum_{i=0}^{|s_a|-1} \sum_{j=0}^{|s_b|-1} (t_{ij})}{|s_a||s_b|} \quad \text{where} \quad t_{ij} = \begin{cases} 1 & \text{if} \quad D[s_a^{start} + i, s_b^{start} + j] \leq 5.0\text{Å} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where 5.0 Å is the threshold distance to define whether two C_α atoms are in contact (i.e., close enough to each other).

Now we can define the CP feature vector $\mathbf{K}^{ab}$ for CP C^{ab} as:

$$\begin{aligned} \mathbf{K}^{ab} = (&CT(s_a, s_b), AS(s_a), SS(s_a), AD(s_a, s_b), SD(s_a, s_b), \\ &\Omega(\mathbf{V}_a, \mathbf{V}_b), ND(\mathbf{V}_a, \mathbf{V}_b), VD(\mathbf{V}_a, \mathbf{V}_b), MD(s_a, s_b), CD(s_a, s_b)). \end{aligned} \tag{14}$$

An example of how to construct a CP feature vector from a real 3D protein structure is given in the supplementary information webpage.

In generating CP feature vectors, when a feature vector has one or more attribute values which are greater than their respective predefined upper bounds (as given in Table 2), it is regarded as an outlier and discarded. These upper bound values are determined empirically. It is observed in our experiments that only about 1% of the CP feature vectors are dropped because they are outliers.

The CP feature vector we use here is sequence-order dependent (referring to attributes 1–5) as opposed to the one used in our previous works. Since the instances of rearrangement of SSEs are rarer than those of insertion and deletion throughout evolution, it will be better to take the sequence-order information into account in our nearest neighbor search. This prevents false matchings of CPs which have very different relative sequence orders of their constituent SSEs. For example, if we have two proteins each having nine SSEs, we will not allow C^{12} from one protein and C^{19} from the other to be matched, because there is little chance that SSE #2 from the first protein is rearranged as SSE #9 in the second protein during evolution. Also, it is not likely that all seven SSEs between SSE #1 and #9 are deleted in the second protein. From our experiments, it is evident that restricting the sequence order of the SSEs (rather than allowing them to float freely in any order) helps improve the accuracy of the scheme. The comparisons are shown in the supplementary information webpage.

3.2.4. Discrete CP feature vector. Now, we encode/discretize the CP feature vector so that it can be represented as a compact 4-byte integer value. In order to do this, we map each attribute value in the feature vector into a discrete number of bins and concatenate all the bits representing these bins into a bit string which can naturally be interpreted as an integer.

The objectives of this encoding are 1) to enable efficient handling of the CP feature vectors and 2) to allow approximate matching of the original CP feature vectors by simply performing exact matching of their discrete versions. The idea of encoding a multidimensional feature vector into a bit string for efficient and effective processing is inspired by that of *VA-File* method (Weber *et al.*, 1998).

The disadvantage of discretization is that there may be false matches (when the attributes values are near the upper and lower boundaries of the same bin) and false mismatches (when the attribute values are near the upper and lower boundaries of the adjacent bins). However, the degree of accuracy provided by the discretization scheme is quite sufficient for our purpose of finding the nearest neighbors of proteins in terms of the number of common CP feature vectors they include—as demonstrated in our experimental results.

The possible ranges for the original space (upper bound) and the discretized space (number of bins and bits) for each attribute are given in Table 2. Some attributes are allocated larger discretized spaces

(i.e., more of bins) than the others because they are found to be relatively more important. (The effect of varying the number of bins for each attribute is shown in the supplementary information webpage.)

As an example, for the closest segment–segment distance (*ND*) attribute, we map an original real number distance value between 0.0 to 64.0 into one of the discrete *bins* numbered 0 to 15 (i.e., 4-bit space). We use simple *equal partition discretization* which is informally defined as follows.

$$bin(Value) = \begin{cases} floor(Value \times NumOfBins/UpperBound) - 1 & \text{if } Value = UpperBound \\ floor(Value \times NumOfBins/UpperBound) & \text{otherwise} \end{cases} \quad (15)$$

For instance, we can calculate the discretized value of a 14.1 Å distance as $floor(14.1 \times 16/64.0) = 3$.

Now, we can define a 27-bit *discrete CP feature vector* $\mathbf{T}$ as follows:

$$\begin{aligned} \mathbf{T}^{ab} = (&bin(CT(s_a, s_b))|bin(AS(s_a))|bin(SS(s_a))|bin(AD(s_a, s_b))|bin(SD(s_a, s_b))| \\ &bin(\Omega(\mathbf{V}_a, \mathbf{V}_b))|bin(ND(\mathbf{V}_a, \mathbf{V}_b))|bin(VD(\mathbf{V}_a, \mathbf{V}_b))| \\ &bin(MD(s_a, s_b))|bin(CD(s_a, s_b))) \end{aligned} \quad (16)$$

where *bin* is the discretization function (Equation 15) and | is the concatenation operator for bit strings.

3.2.5. CPset. Now, we can encode an entire 3D protein structure as a set of discrete CP feature vectors it contains. We call this set a *CPset* and denote it as $\mathcal{CPS}$. Set $\mathcal{CPS}_P$ of protein structure P with $|S|$ SSEs can be defined as

$$\mathcal{CPS}_P = \{\mathbf{T}^{12}, \mathbf{T}^{13}, \ldots, \mathbf{T}^{1|S|}, \mathbf{T}^{23}, \ldots, \mathbf{T}^{(|S|-1)|S|}\} \quad (17)$$

where $\mathbf{T}^{\mathbf{ij}}$ (where $1 \leq i \leq (|S| - 1)$ and $(i + 1) \leq j \leq |S|$) is a discrete CP feature vector.

The cardinality of a CPset with $|S|$ SSEs is at most $|S|(|S| - 1)/2 = O(|S|^2)$. (There may be some outlier CPs which are excluded from the CPset.)

We sort the discrete CP feature vectors (which can be regarded as integers) in the CPset in ascending order to enable linear-time comparison of them in the later classification step. Sorting alters the original order of these discrete feature vectors in the CPset. However, since the discretized attributes *AS* (position of first SSE in AA sequence), *SS* (position of first SSE in SSE sequence), *AD* (AA position difference between first and second SSEs), and *SD* (SSE position difference between first and second SSEs) are stored in the discrete CP feature vector, the positions and relative order of the original CPs in the distance matrix can still be roughly determined.

Both PA and CPset are compact and efficient means of encoding a 3D protein structure. The average sizes of a PA and a CPset are 160 and 782 bytes, respectively, whereas the average size of an original PDB format file (3D coordinates only, without any annotation) is about 261 KB (261,000 bytes).

4. PROTCLASS METHOD

In this section, we discuss the *preprocessing step* and the *querying (classification) step* of the ProtClass method.

In the preprocessing step, the system first generates the database of protein abstracts (PAs) and the database of sets of discrete contact pattern feature vectors (CPsets) from the training data (the database of 3D protein structures with known class labels). Then, from the PA database, it computes the PA distance threshold parameters for each class by all-against-all comparison of the PAs in the given class.

In the querying step, the system generates the query's PA and CPset in the same manner. Then, in the first filtering substep, it prunes away the unpromising answers by comparing the PA of the query against those of the database proteins, using the PA distance threshold parameters learned in the preprocessing step. Then, in the refinement substep, the system conducts a nearest neighbor search of the query's CPset against the remaining database proteins' CPsets and returns the class label(s) of the protein(s) that are best matched.

An overview of the method is illustrated in Fig. 2. The algorithmic details of the preprocessing and querying steps are described in the following two subsections.

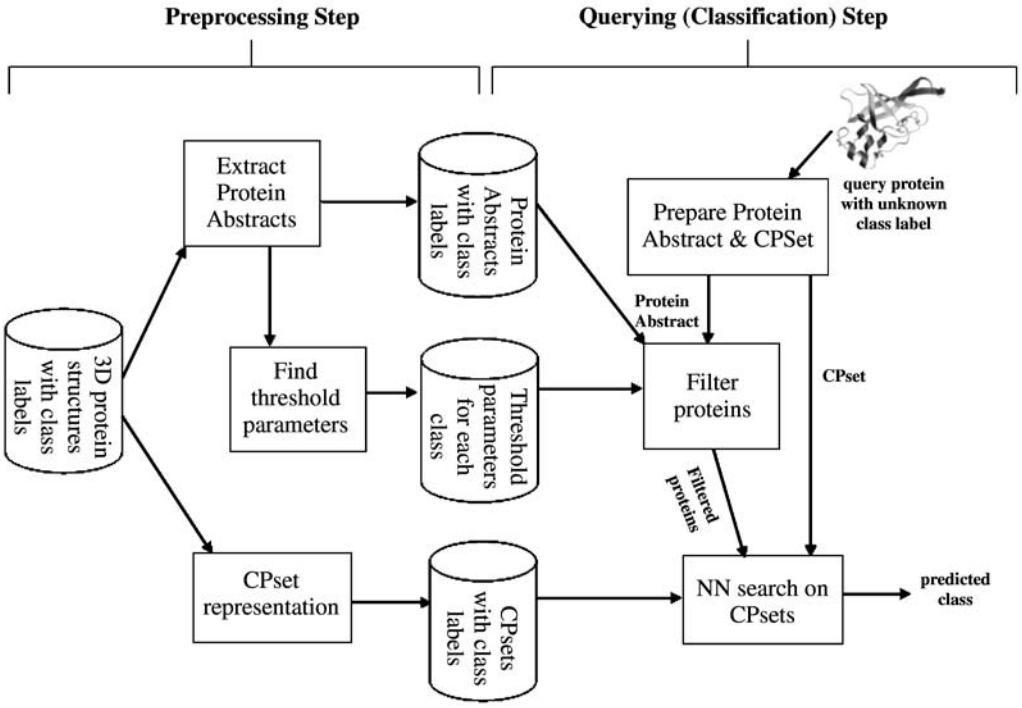


FIG. 2. Overview of ProtClass method.

4.1. Preprocessing algorithm

Given a database of protein structures with known class labels, we can generate the databases of PAs, PA distance thresholds, and CPsets using the preprocessing algorithm shown in Fig. 3. The symbols used in the algorithm have the same meanings as in their previous definitions unless redefined otherwise.

In the preprocessing algorithm, for each distinct class, we generate the PAs and CPsets for the proteins belonging to this class and store them in the database (lines 4–10). Again, for each class, we calculate the farthest PA attribute distances $max\delta_b$ that are exhibited in the PA pairs belonging to this class (lines 15–20). For each PA attribute, its farthest distance value is multiplied with a *CoverageFactor* to obtain the distance threshold value of this attribute (line 23).

The empirically determined value, $CoverageFactor = 0.80$, is used in our implementation. The objective of setting the threshold distances is to ensure that at least one "nearest" protein belonging to the same class as a given query protein will pass the filtering step (see next subsection), whilst the dissimilar proteins are discarded straight away. It means that for each PA attribute, we expect to find the nearest neighbor(s) to a query protein within 80% of the distance of the two farthest proteins (with respect to this attribute) belonging to this particular class in the existing database. If the number of trained proteins in a class is too few (say less than 20), this condition may not be always true. We use the default threshold values ($\delta_{|A|} = 0.4$, $\delta_{|S|} = 0.4$, $\delta_{SL} = 0.3$, $\delta_{HL} = 0.2$, $\delta_{HN} = 0.3$, $\delta_S = 0.5$) in this case.

4.2. Querying algorithm

The algorithm in Fig. 4 describes how unpromising answers with respect to a given query can be filtered out using PAs and how the class of the query protein can be predicted by using the nearest neighbor search based on PA similarity and CPset similarity.

In the *filtering step*, the algorithm first filters out the unpromising answers using the PA of the query and those of the database proteins (lines 9–10). For a protein that passes the filtering test, the matching score of its PA to the query PA ($PAScore$) is calculated from the Euclidean distance of this PA pair (line 11). (The maximum possible Euclidean distance between two PAs is $\sqrt{6}$, since a PA has six attributes whose values are in the range 0 to 1.)

Input: Protein structure database $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$ where $\mathcal{D}_i$ $(1 \leq i \leq n)$
is a set of protein structures with class label i

Output: (1) PA database $\mathcal{PA} = \{\mathcal{PA}_1, \mathcal{PA}_2, \ldots, \mathcal{PA}_n\}$ where $\mathcal{PA}_i$ $(1 \leq i \leq n)$
is a set of Protein Abstracts for proteins with class label i

Output: (2) PA distance threshold database $\delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$ where δ_i $(1 \leq i \leq n)$
is a set of PA distance thresholds class i

Output: (3) CPset database $\mathcal{CPS} = \{\mathcal{CPS}_1, \mathcal{CPS}_2, \ldots, \mathcal{CPS}_n\}$ where $\mathcal{CPS}_i$ $(1 \leq i \leq n)$
is a set of CPsets for proteins with class label i

1. $\mathcal{PA} = \phi$, $\mathcal{CPS} = \phi$, $\delta = \phi$
2. for $i = 1$ to n do
3. $\quad \mathcal{PA}_i = \phi$, $\delta_i = \phi$, $\mathcal{CPS}_i = \phi$
4. $\quad$ for $j = 1$ to $|\mathcal{D}_i|$ do
5. $\quad\quad$ Let $\mathcal{D}_{ij}$ $(1 \leq j \leq |\mathcal{D}_i|)$ be an individual protein structure in $\mathcal{D}_i$
6. $\quad\quad$ $\mathcal{PA}_{ij} =$ **GeneratePA**$(\mathcal{D}_{ij})$ /* See Sub-section 3.1 */
7. $\quad\quad$ $\mathcal{PA}_i = \mathcal{PA}_i \cup \mathcal{PA}_{ij}$
8. $\quad\quad$ $\mathcal{CPS}_{ij} =$ **GenerateCPset**$(\mathcal{D}_{ij})$ /* See Sub-section 3.2 */
9. $\quad\quad$ **Sort**$(\mathcal{CPS}_{ij})$ /* Sort in ascending order */
10. $\quad\quad$ $\mathcal{CPS}_i = \mathcal{CPS}_i \cup \mathcal{CPS}_{ij}$
11. $\quad$ end for
12. $\quad$ for each $b \in \{|A|, |S|, SL, HL, HN, S\}$ do /* b is a PA attribute */
13. $\quad\quad$ $max\delta_b = 0$
14. $\quad$ end for
15. $\quad$ for $j = 1$ to $|\mathcal{D}_i|$ do
16. $\quad\quad$ for $k = 1$ to $|\mathcal{D}_i|$ do
17. $\quad\quad\quad$ for each $b \in \{|A|, |S|, SL, HL, HN, S\}$ do
18. $\quad\quad\quad\quad$ if $(max\delta_b < \Delta_b(\mathcal{PA}_{ij}, \mathcal{PA}_{ik}))$ then $max\delta_b = \Delta_b(\mathcal{PA}_{ij}, \mathcal{PA}_{ik})$ /* Eq. 4 and 5 */
19. $\quad\quad\quad$ end for
20. $\quad\quad$ end for
21. $\quad$ end for
22. $\quad$ for each $b \in \{|A|, |S|, SL, HL, HN, S\}$ do
23. $\quad\quad$ $\delta_{ib} = max\delta_b \times CoverageFactor$
24. $\quad\quad$ $\delta_i = \delta_i \cup \delta_{ib}$
25. $\quad$ end for
26. $\quad$ $\mathcal{PA} = \mathcal{PA} \cup \mathcal{PA}_i$, $\mathcal{CPS} = \mathcal{CPS} \cup \mathcal{CPS}_i$, $\delta = \delta \cup \delta_i$
27. end for
28. return $(\mathcal{PA}, \delta, \mathcal{CPS})$

FIG. 3. ProtClass preprocessing algorithm.

Input: (1) Query protein structure Q

Input: (2) $\mathcal{PA}, \delta, \mathcal{CPS}$ /* See their definitions in above algorithm Figure 3. */

Output (1): Most possible class $MaxClass$ for Q and its score $MaxClassScore$

Output (2): Most possible proteins $MaxProtein$ and their respective scores $MaxScore$ arrays for all distinct classes /* optional */

1. $PAQ = \textbf{GeneratePA}(Q)$ /* See Sub-section 3.1 */
2. $CPSQ = \textbf{GenerateCPset}(Q)$ /* See Sub-section 3.2 */
3. $\textbf{Sort}(CPSQ)$ /* Sort in ascending order */
4. Let $CPSQ = \{\mathbf{T}_{Q1}, \mathbf{T}_{Q2}, \ldots, \mathbf{T}_{Q|CPSQ|}\}$ where $\mathbf{T}_{Qx}$ $(1 \le x \le |CPSQ|)$ is a discrete CP feature vector of Q.
5. $MaxClassScore = 0$, $MaxClass = 0$
6. for $i = 1$ to n do /* n is the number of distinct classes */
7. $\quad MaxScore_i = 0$, $MaxProtein_i = 0$
8. $\quad$ for $j = 1$ to $|\mathcal{PA}_i|$ do /* $|\mathcal{PA}_i|$ is the number of trained proteins in class i */
9. $\quad\quad$ if ($Similar(PAQ, \mathcal{PA}_{ij}, \delta_i) ==$ false) then /* See Eq. 6 */
10. $\quad\quad\quad$ continue; /* filter test failed; skip this protein and process next one */
11. $\quad\quad PAScore = (\sqrt{6} - PADist(PAQ, \mathcal{PA}_{ij}, \delta_i))/\sqrt{6}$ /* See Eq. 7 */
12. $\quad\quad$ Let $\mathcal{CPS}_{ij} = \{\mathbf{T}_{P1}, \mathbf{T}_{P2}, \ldots, \mathbf{T}_{P|\mathcal{CPS}_{ij}|}\}$ where $\mathbf{T}_{Py}$ $(1 \le y \le |\mathcal{CPS}_{ij}|)$ is a discrete CP feature vector of $\mathcal{CPS}_{ij}$.
13. $\quad\quad x = 1$, $y = 1$, $MatchCount = 0$
14. $\quad\quad$ while $(x \le |CPSQ|)$ and $(y \le |\mathcal{CPS}_{ij}|)$ do
15. $\quad\quad\quad$ if ($\mathbf{T}_{Qx} == \mathbf{T}_{Py}$) then $MatchCount$ ++, x ++, y ++
16. $\quad\quad\quad$ else if ($\mathbf{T}_{Qx} < \mathbf{T}_{Py}$) then x ++
17. $\quad\quad\quad$ else y ++
18. $\quad\quad$ end while
19. $\quad\quad CPSetScore = MatchCount/|CPSQ|$ /* Normalize */
20. $\quad\quad FinalScore = PAScore \times CPSetScore$
21. $\quad\quad$ if ($FinalScore > MaxScore_i$) then $MaxScore_i = FinalScore$, $MaxProtein_i = j$
22. $\quad$ end for
23. $\quad$ if ($MaxScore_i > MaxClassScore$) then $MaxClassScore = MaxScore_i$, $MaxClass = i$
24. end for
25. return $((MaxClass, MaxClassScore), (MaxProtein, MaxScore))$

FIG. 4. ProtClass querying (classification) algorithm.

Our experimental results show that an average of 71% of the database proteins are discarded in the filtering step, as they are not close enough to the query. An average of 38% of the classes are entirely discarded. It should be noted that the proteins that are in the same class as the query but not similar enough (the distantly related ones) are also discarded. But the similar ones in the correct class pass the test. So, there is no chance of discarding the correct class as a whole.

In the second *refinement step*, the matching score of the protein's CPset to that of the query ($CPsetScore$) is calculated by assembling (merging) their constituent discrete CP feature vectors (lines 13–18). Although it involves a pairwise comparison, it is very much faster than the detailed structural alignment procedures, because it is only a linear time algorithm (in terms of the number of CPs) which merely requires the comparison of integer values. The final score of the protein with respect to the query is calculated by taking both the PA matching score (*PAScore*) and the CPset matching score (*CPsetScore*) into account (line 20). So, the similarity metric for our nearest neighbor search depends on both types of scores. Finally we return the class label of the protein which is nearest to the query (in terms of the final score) and, optionally, other information such as the best scoring protein for each class, etc. (line 25).

5. RESULTS AND DISCUSSIONS

In order to assess the accuracy and efficiency of the proposed ProtClass scheme, we test it on a medium size dataset with 600 protein structures in the experimental setup mentioned below. We compare ProtClass against DaliLite (Holm and Park, 2000) (the stand-alone version of DALI), SGM (Røgen and Fain, 2003), and CPMine (Aung and Tan, 2004a) using their default settings in the same experimental setup. When running DaliLite, we use its database search option rather than its pairwise alignment option.

5.1. Experimental setup

We use the *ASTRAL* dataset (Brenner *et al.*, 2000) that contains proteins with less than 40% sequence homology. (ASTRAL provides the PDB-style 3D coordinate files for the protein domains as defined by SCOP.) From this dataset, we choose 15 *Folds* (according to SCOP designation) each with 40 member proteins. (For Folds with more than 40 members, we randomly select 40 from them.) Thus, we have a pool of $15 \times 40 = 600$ protein structures whose class labels (Folds) are known. We use these 15 Folds as our target structural classes.

We conduct our experiment using a *10-fold cross validation* strategy. We split each Fold into 10 partitions each having four protein structures. We conduct 10 subexperiments in each of which we build the *testing dataset* by choosing a partition from each of the Folds and combining them together. In this way, we have a testing dataset consisting of 60 proteins for each subexperiment. The remaining 540 proteins are used as the *training dataset*. The training dataset is made up of 36 proteins from each of 15 Folds.

In other words, in each subexperiment, we have a database of 540 protein structures whose Folds are already known and a set of 60 query protein structures whose Folds are to be predicted. We then validate the predicted Folds of the query proteins against their actual Folds as designated by SCOP.

5.2. Accuracy

In the above-mentioned manner, 10 subexperiments are conducted by using the different testing sets and training sets in each cycle. Then, we consolidate the results from 10 subexperiments and calculate the average accuracy of the scheme. We look at the top-three scoring Folds (which are 20% of the total number of distinct Folds) and examine whether they are actually the correct classifications. The results are shown in Table 3. Column "Top 1" shows the accuracy of the scheme if only the topmost scorer is examined, and Column "Top 2" shows the accuracy if both the top-one and top-two scorers are examined, etc. The accuracy results of DaliLite, SGM, and CPMine are also shown in Table 3.

With a large number of possible classes (15 SCOP Folds in this case), it may be more useful to report a few candidate classes rather than to report a single but incorrect class—as long as the number of reported classes is only a small fraction of all the possible classes (20% in our case). The user can manage these candidate classes according to his/her own requirement. For example, if only moderate accuracy in

TABLE 3. EXPERIMENTAL RESULTS ON 15 DISTINCT FOLDS

		Average percentage of correct classifications											
		DaliLite			*SGM*			*ProtClass*			*CPMine*		
Fold	*Class*	*Top1*	*Top2*	*Top3*	*Top1*	*Top2*	*Top3*	*Top1*	*Top2*	*Top3*	*Top1*	*Top2*	*Top3*
46688	All-α	92.5	95.0	95.0	80.0	95.0	97.5	92.5	97.5	100.0	80.0	85.0	92.5
47472	All-α	100.0	100.0	100.0	80.0	90.0	97.5	100.0	100.0	100.0	47.5	97.5	100.0
48370	All-α	100.0	100.0	100.0	97.5	100.0	100.0	100.0	100.0	100.0	50.0	72.5	85.0
48725	All-β	100.0	100.0	100.0	100.0	100.0	100.0	95.0	97.5	100.0	72.5	97.5	100.0
50198	All-β	100.0	100.0	100.0	82.5	92.5	95.0	90.0	92.5	100.0	30.0	62.5	85.0
51350	α/β	100.0	100.0	100.0	77.5	92.5	97.5	97.5	97.5	100.0	47.5	65.0	77.5
51734	α/β	100.0	100.0	100.0	72.5	95.0	100.0	97.5	100.0	100.0	52.5	85.0	87.5
51904	α/β	100.0	100.0	100.0	97.5	97.5	100.0	95.0	97.5	97.5	32.5	55.0	60.0
52171	α/β	95.0	100.0	100.0	65.0	82.5	82.5	85.0	95.0	95.0	42.5	72.5	95.0
52539	α/β	100.0	100.0	100.0	50.0	72.5	85.0	70.0	92.5	100.0	32.5	60.0	82.5
52832	α/β	100.0	100.0	100.0	90.0	92.5	95.0	95.0	97.5	100.0	25.0	37.5	62.5
53066	α/β	100.0	100.0	100.0	67.5	80.0	92.5	80.0	95.0	97.5	30.0	37.5	47.5
53473	α/β	100.0	100.0	100.0	77.5	92.5	92.5	92.5	97.5	100.0	47.5	60.0	70.0
54235	$\alpha+\beta$	100.0	100.0	100.0	87.5	92.5	100.0	82.5	97.5	100.0	50.0	70.0	85.0
54861	$\alpha+\beta$	100.0	100.0	100.0	85.0	97.5	100.0	85.0	92.5	100.0	32.5	67.5	90.0
Overall		99.2	99.7	99.7	80.7	91.5	95.7	90.5	96.7	99.3	44.8	68.3	81.3

classification is required, the user can just take the topmost scoring class as the answer. If high accuracy is needed, manual inspections or detailed structural alignments can be done on the top scoring members of the top-three scoring classes. This strategy of reporting more than one class as possible candidates is also advocated by the authors of SGM (Røgen and Fain, 2003).

From the experiments, it is observed that ProtClass performs quite accurately. Overall, it offers an average accuracy of 99.33% if we take the top-three scorers into account and 90.50% if we take only the topmost scorer into account.

It gives perfect results on certain Folds such as 47472 and 48370 with 100% accuracy even with the topmost scorer. It performs fairly well on certain Folds such as 52171 (95% accuracy with the top-three scorers and 85% with the topmost scorer).

As expected, DaliLite exhibits more accuracy than does ProtClass. Its high accuracy (99.7% with the topmost and 99.2% with top three) can be attributed to its detailed structural alignment mechanism. But, on the other hand, this approach is extremely slow (see Subsection 5.3). We can achieve the same high accuracy as DaliLite while reducing the running time by employing ProtClass as a rapid query preprocessor. We can run the filtering step of ProtClass before running DaliLite itself. The filtering step filters out about 71% of all the database proteins as irrelevant. It is observed that the proteins that are relevant to the final answer are always retained in the remaining 29% to be processed by DaliLite in the next step. Therefore we do not miss out on anything, but can reduce the total running time by about 73%. This means 370% improvement in speed. (The time reduced may not always be proportional to the number of proteins filtered out, because different proteins incur different alignment costs with respect to the query.)

The overall accuracy of ProtClass is slightly better than that of SGM. Out of 15 Folds tested, ProtClass performs better than SGM in 9 Folds, equally in 5 Folds, and poorer in 1 Fold (according to the top-three scorer results). We can observe some similarities between the result of ProtClass and that of SGM. For example, both methods give perfect results for Folds 48370, 48725, 51734, etc., but give poor topmost scorer results for Fold 52439.

In comparison with CPMine, ProtClass is found to be much more accurate. This is because CPMine is a fingerprint-based comparison approach without any filters. Matching a query against the fingerprints of the classes (Folds in this case) can introduce both false positives and false negatives, because the fingerprints sometimes cannot represent their respective classes uniquely and unfailingly. In ProtClass, we do not use such a fingerprinting mechanism, thus reducing the possibility of false classifications due to misrepresentation.

TABLE 4. AVERAGE RUNNING TIMES FOR 60 QUERIES ON 540 PROTEINS: (A) FOR FOUR METHODS, (B) BREAKDOWN OF COSTS FOR PROTCLASS

(a)

	Average time elapsed (in seconds)				*Average time on one protein/query (in seconds)*			
Description	*DaliLite*	*SGM*	*ProtClass*	*CPMine*	*DaliLite*	*SGM*	*ProtClass*	*CPMine*
Preprocessing (540 proteins)	1,080	4,297	224	227	2.00	7.96	0.41	0.42
Querying (60 proteins)	163,466	478	30	26	2,724.43	7.97	0.50	0.43
Total	164,546	4,775	254	253				

(b)

Description	*Average time elapsed (in seconds)*	*Average time on one protein/query (in seconds)*
Preprocessing (540 proteins)		
• Running Stride	211.53	0.39
• Generating PAs, CPsets, and PA distance threshold databases	12.65	0.02
Total	224.18	
Querying (60 queries)		
• Running Stride	24.05	0.40
• Preparing queries (PA and CPset)	1.31	0.02
• Filtering step	1.94	0.03
• Refinement step	2.85	0.05
Total	30.15	

5.3. Speed

It is observed that the proposed ProtClass scheme works very efficiently. All the experiments are done on Sun Ultra Sparc II with two 480 MHz CPUs and 4 GB main memory, running Sun OS 5.7. The time statistics are shown in Table 4(a). It shows the time taken to run one cycle of a subexperiment (preprocessing on 540 proteins, and querying with 60 proteins) by each method. It is averaged out from the times taken by the 10 subexperiments in 10-fold cross validation.

Although DaliLite is more accurate than ProtClass, it is found to be about 640 times slower! It takes an average of about 1 day and 21 hours for querying of 60 proteins on the database of 540 proteins. In fact, it may be impractical to use DaliLite for a real-time classification task involving a large database on an average stand-alone machine. In the previous subsection, we have already discussed how we can improve the running time of DaliLite by using ProtClass as a rapid preprocessor.

In comparison with SGM, it is observed that ProtClass is about 18 times faster. This is because SGM involves a large number of floating point operations to calculate the Gauss integrals. In contrast, ProtClass mainly performs integer and bitwise operations. CPMine's running time is about the same as that of ProtClass. They both use the similar integer and bit operations.

From the experiment, it is also observed that the running time of ProtClass, both for preprocessing and querying steps, is overwhelmed by that of the Stride external algorithm used to generate the SSE information. The breakdown of the various time costs in both preprocessing and querying steps are shown in Table 4(b).

5.4. Importance of filter and refine steps

With the hope to further improve the accuracy of the scheme, one may be tempted to drop the filtering step and run only the relatively more detailed refinement step on every database protein. But, unfortunately, this does not work. The accuracy of the system is degraded substantially if filtering is not carried out. This is because the filtering step can prune away a lot of potential false positive proteins whose CPsets are similar to those of the query, but whose PAs are not. Our experimental results show that both filtering and refinement steps are indispensable. Figure 5(a) shows the overall accuracy of the system when both steps are included, and for each step separately.

5.5. Importance of PA attributes

In order to assess the importance of the six attributes in a PA, we drop each attribute at a time and rerun the experiment as described in Subsection 5.1. It is observed that all the attributes are more or less important on their own, because dropping any of them reduces the accuracy of the scheme—as shown in Fig. 5(b). Although dropping the *HN* attribute gives the same accuracies for the top one and two scoring Folds as the original scheme, its accuracy for the top-three scorers is lower. Dropping *SL* and *S* attributes gives nearly the same accuracy for the top two and three scorers as the original scheme, but their topmost scorer accuracies are lower.

5.6. Importance of CP feature vector attributes

Similarly, in order to evaluate the importance of the 10 attributes in a CP feature vector, we exclude each attribute in turn from the experiment as described in Subsection 5.1. Again, it is found out that all the attributes are more or less important, because dropping any of them degrades the accuracy of the scheme—as shown in Fig. 5(c). Although all the attributes are important, some attributes—such as CT (contact pattern type) and Ω (torsion angle)—are found to be more important than the others, as the exclusion of these attribute affects the accuracy of the system more seriously.

5.7. Effect of proportion of training and testing data

In our 10-fold cross validation experiment on 600 proteins, we use 540 (90%) of them as the training data and the remaining 60 as the testing data in each cycle. In order to explore the effect of the proportion of the training and testing data, we change it variously and observe the changes in the accuracy of the scheme. First, we conduct the leave-one-out test by using 15 proteins as the testing data (one from each distinct Fold) and the remaining 585 as the training data and repeat the experiment 40 times. In Fig. 6(a), the label on the x-axis "97.5%" means that the percentage of the training proteins is 97.5% of the total proteins (i.e., 585 out of 600). The label "(39/40)" means that 39 out of 40 proteins in each distinct Fold are used as the training data. The other test cases are with 75%, 50%, and 25% training data, respectively. All these are compared against our default test with 90% training data as shown in the figure.

Similar experiments are also conducted on SGM whose results are also shown in Fig. 6(a). We exclude DaliLite and CPMine from our experiment, because the former takes a very long time to run and the latter is clearly inferior to ProtClass.

As expected, the accuracy of the scheme gradually delines with the reduction of the training data percentage. However, there are no steep slopes in the curves. In the worst case with 25% training data, ProtClass can still provide accuracies of 77.4%, 88.0%, and 93.2% for the topmost, top-two, and top-three answers, respectively. We can also observe a similar trend of declining accuracy in the curves of SGM. In fact, this is a general phenomenon for all classification systems—the more the training data, the better the accuracy.

5.8. Effect of class size

In our experiment, we use classes (SCOP Folds) with 40 members each, which can be considered as relatively big ones. In order to assess the performance of the scheme on various sizes of Folds, we run multiple tests on various Fold sizes. In the first test, we take Folds with at least two members from the

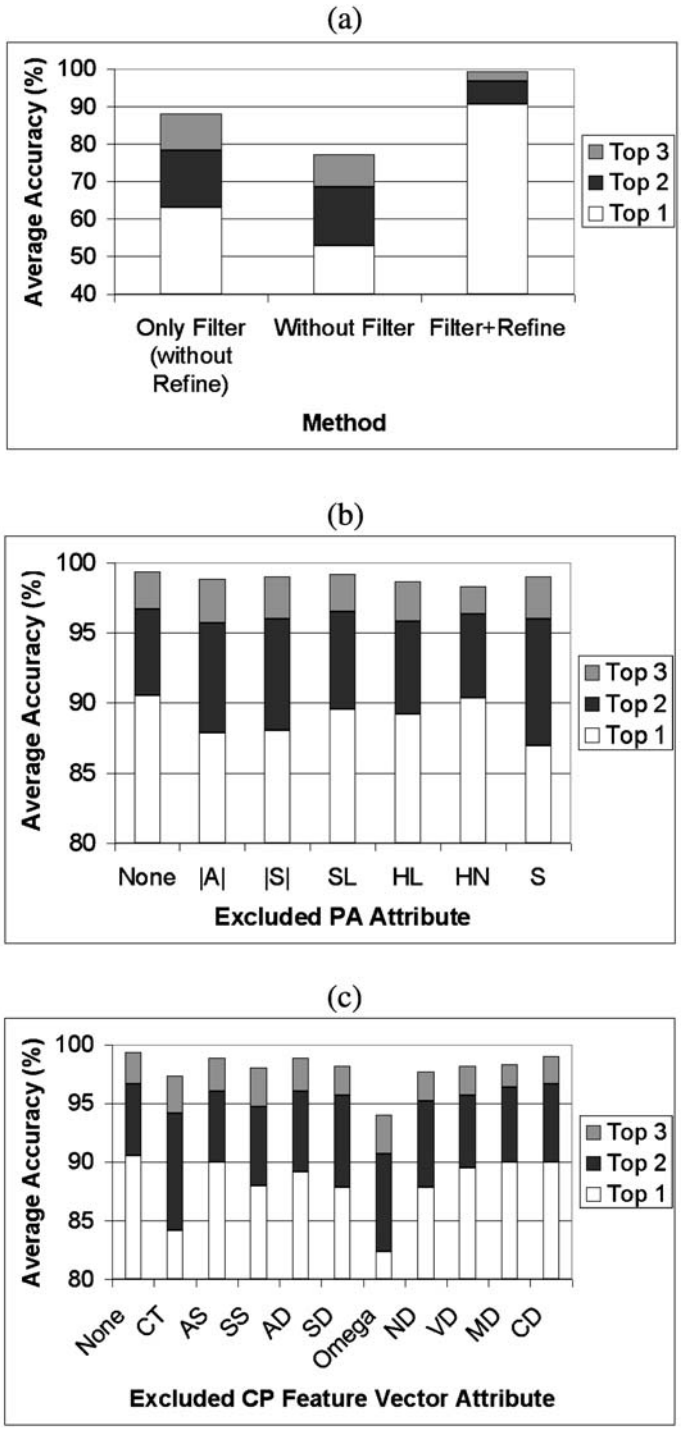
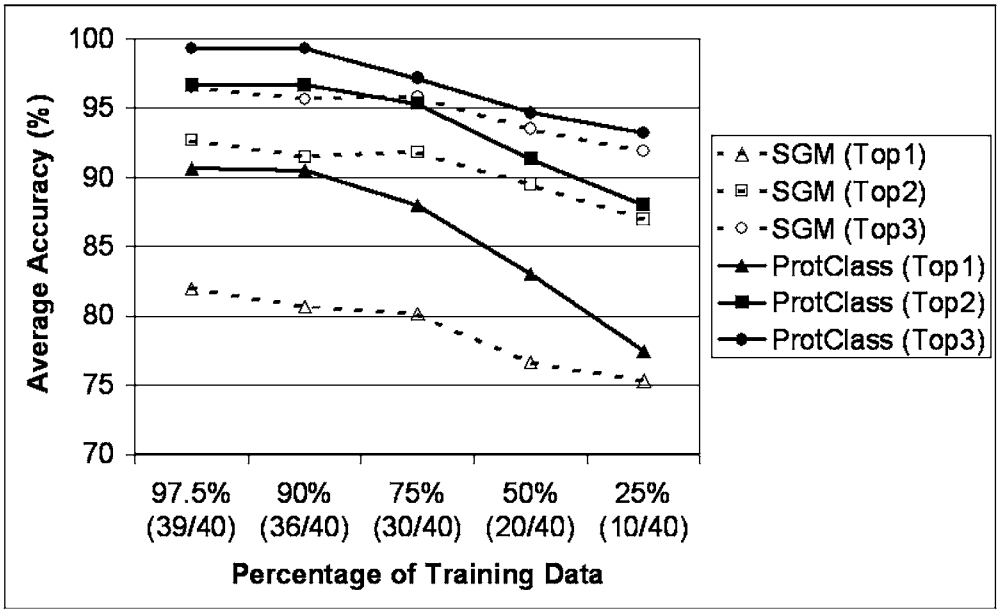


FIG. 5. Importance of (**a**) filter and refine steps, (**b**) each PA attribute, and (**c**) each CP feature vector attribute.

ASTRAL dataset with less than 40% sequence homology. (For Folds with more than two members, we randomly choose two from them.) There are 350 such Folds, thus yielding a dataset with $(350 \times 2 = 700)$ proteins. Then we run two test cycles, each with 350 training proteins and 350 testing proteins. The other test cases are for Folds with 5, 10, and 15 members, etc., up to 50 members. The results are shown in Fig. 6(b). Similar experiments are also conducted on SGM.

(a)



(b)

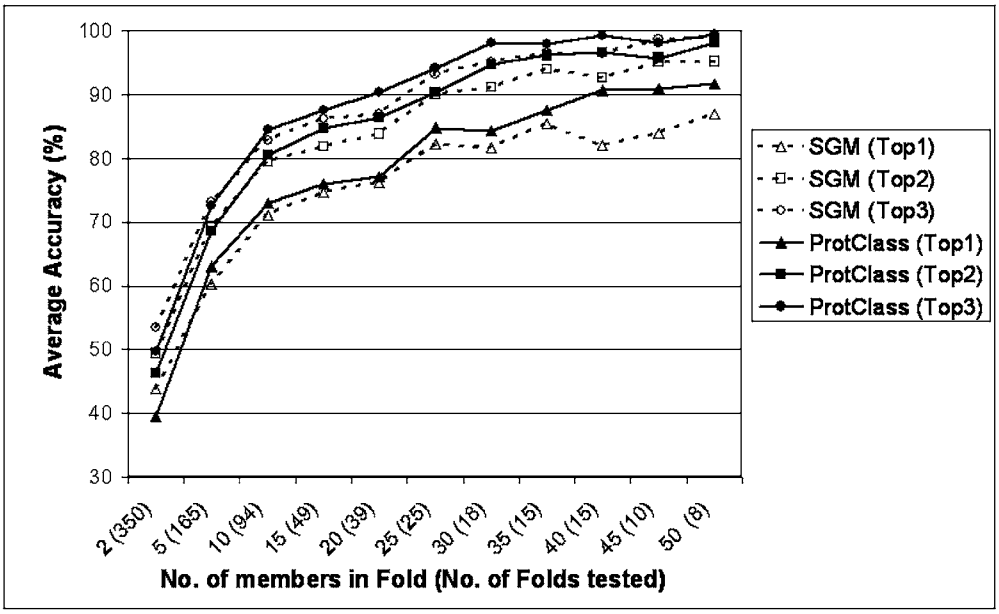


FIG. 6. Effect of (**a**) percentage of training data and (**b**) number of members in each distinct Fold.

ProtClass can provide only 50% accuracy (with the top-three scorers) for the Folds with two members. However, given a very large number of possible Folds (350), this 50% accuracy is not trivial. We can observe the trend of accuracy improvement with the increased number of members in the Folds. ProtClass can provide a reasonable accuracy of at least 80% for the Folds with 10 or more members when the top-three scorers are taken into account, and 25 or more members when only the topmost scorer is taken into account. We can also see a similar trend of accuracy improvement in the curves of SGM. The accuracy of SGM is better than ProtClass for two-member Folds, but the latter is generally better for the rest of the Folds. It is interesting to observe a high degree of correlation between the curves of ProtClass and those of SGM.

6. CONCLUSION

In this paper, we have presented a new automatic scheme for protein structure classification. Our system is a dedicated classifier that does not require a costly structural alignment process. The experimental results show that our method is accurate and very efficient.

As future work, we can further assess the general behavior of our scheme by trying it with a greater number of training and testing proteins taken from a greater number of structural classes. We can improve the accuracy and efficiency of classification even further by introducing a better filtration scheme, a better CPset feature vector representation scheme, a better discretization scheme, etc.

Finally, we believe our scheme can become a very useful automatic tool for rapid and accurate protein structure classification in the age of very large protein structure databases.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments and valuable suggestions. We also thank the authors of Stride (Frishman and Argos, 1995), DaliLite (Holm and Park, 2000), and SGM (Røgen and Fain, 2003) for making the source codes of their respective programs available to us.

REFERENCES

Aung, Z., and Tan, K.L. 2004a. Automatic protein structure classification through structural fingerprinting. *Proc. 4th IEEE Symp. on Bioinformatics and Bioengineering (BIBE '04)*, 508–515.

Aung, Z., and Tan, K.L. 2004b. Rapid 3D protein structure database searching using information retrieval techniques. *Bioinformatics* 20, 1045–1052.

Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P. 2000. The Protein Data Bank. *Nucl. Acids Res.* 28, 235–242.

Brenner, S.E., Koehl, P., and Levitt, M. 2000. The ASTRAL compendium for sequence and structure analysis. *Nucl. Acids Res.* 28, 254–256.

Camoglu, O., Kahveci, T., and Singh, A.K. 2003. PSI: Indexing protein structures for fast similarity search. *Bioinformatics* 19, 81i–83i.

Caprara, A., Carr, R., Istrail, S., Lancia, G., and Walenz, B. 2004. 1001 optimal PDB structure alignments: Integer programming methods for finding the maximum contact map overlap. *J. Comp. Biol.* 11, 27–52.

Chinnasamy, A., Sung, W.K., and Mittal, A. 2004. Protein structure and fold prediction using tree-augmented Bayesian classifier. *Proc. 9th Pacific Symp. on Biocomputing (PSB '04)*.

Ding, C.H.Q., and Dubchak, I. 2001. Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics* 17, 349–358.

Eidhammer, I., Jonassen, I., and Taylor, W.R. 2000. Protein structure comparison and structure patterns. *J. Comp. Biol.* 7, 685–716.

Fischer, D., Tsai, C.J., and Nussinov, R., and Wolfson, H.J. 1995. A 3D sequence-independent representation of the Protein Data Bank. *Protein Eng.* 8, 981–997.

Frishman, D., and Argos, P. 1995. Knowledge-based secondary structure assignment. *Proteins Struct. Funct. Genet.* 23, 566–579.

Gibrat, J.F., Madej, T., and Bryant, H. 1996. Surprising similarities in structure comparison. *Curr. Opin. Struct. Biol.* 6, 377–385.

Gusfield, D. 1997. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, London.

Holm, L., and Park, J. 2000. DaliLite workbench for protein structure comparison. *Bioinformatics* 16, 566–567.

Holm, L., and Sander, C. 1993. Protein structure comparison by alignment of distance matrices. *J. Mol. Biol.* 233, 123–138.

Huan, J., Wang, W., Washington, A., Prins, J., and Tropsha, A. 2004. Accurate classification of protein structural families using coherent subgraph analysis. *Proc. 9th Pacific Symp. on Biocomputing (PSB '04)*.

Hubbard, T.J.P., Ailey, B., Brenner, S.E., Murzin, A.G., and Chothia, C. 1997. SCOP: A structural classification of proteins database. *Nucl. Acids Res.* 25, 236–239.

Lancia, G., and Istrail, S. 2004. Protein structure comparison: Algorithms and applications. *In* Guerra, C., and Istrail, S., eds., *Mathematical Methods for Protein Structure Analysis and Design*, 1–33, Springer-Verlag, Heidelberg.

Orengo, C.A., Jones, D.T., and Thornton, J.M. 2003. *Bioinformatics: Genes, Proteins and Computers*, BIOS Scientific, Oxford.

Orengo, C.A., Michie, A.D., Jones, S., Jones, D.T., Swindells, M.B., and Thornton, J.M. 1997. CATH: A hierarchic classification of protein domain structures. *Structure* 5, 1093–1108.

Røgen, P., and Fain, B. 2003. Automatic classification of protein structure by using Gauss integrals. *Proc. Natl. Acad. Sci. USA* 100, 119–124.

Sfyrakis, K. 2004. Geometrical transformations. *http://lcvmwww.epfl.ch/∼kostas/GeometricalTransformations.htm#Anchor-New-49572*.

Shindyalov, I.N., and Bourne, P.E. 1998. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Eng.* 11, 739–747.

Singh, A.P., and Brutlag, D.L. 1997. Hierarchical protein structure superposition using both secondary structure and atomic representations. *Proc. 5th Int. Conf. on Intelligent Systems for Molecular Biology (ISMB '97)*, 284–293.

Sunday, D. 2004. Distance between lines and segments with their closest point of approach. *http://softsurfer.com/Archive/algorithm_0106/algorithm_0106.htm*.

Weber, R., Schek, H.J., and Blott, S. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proc. 24th Int. Conf. on Very Large Data Bases (VLDB '98)*, 194–205.

Address correspondence to:
Zeyar Aung
Department of Computer Science
National University of Singapore
3 Science Drive 2
Singapore 117543

E-mail: zeyaraun@comp.nus.edu.sg