An Efficient Index-based Protein Structure Database Searching Method

Zeyar Aung[1] Wei Fu[1] Kian-Lee Tan[1,2]

[1]*Department Computer Science*
National University of Singapore
3 Science Drive 2, Singapore 117543
{*zeyaraun, fuwei, tankl*}*@comp.nus.edu.sg*

[2]*Genome Institute of Singapore*
1 Science Park Road, The Capricorn
#05-01, Singapore 117528
gistankl@nus.edu.sg

Abstract

In this paper, we present a novel indexing method called ProtDex to facilitate fast searching in 3-dimensional protein structure database. In ProtDex, we first build an index on the representative properties of all proteins in the database. When evaluating a query, with the help of the index, we filter out a small candidate list of proteins. Then, we can either directly report them, with their respective rankings, to the user, or do the expensive actual alignments on them upon user's request. Preliminary experimental results show that our solution is up to 16 times faster than the popular DALI method for database searching task (without actual alignments), while its overall accuracy is only slightly inferior to that of DALI. The software is available upon request by sending emails to the authors.

1. Introduction

Advances in laboratory methods to determine the structures of bio-molecules have led to a rapid growth in the sizes of the protein structure databases such as *PDB* [2]. For example, PDB stored only about 1,000 structures in 1992, but as of September 2002 it stores over 18,000 structures. As a result, faster tools for structural comparison and database searching become essential. Structural data are 3-dimensional in nature. Some huge structures are composed of thousands of atoms. Unlike 1-dimensional sequence comparison, it is much more complex and computationally expensive to compare two structures to determine their similarity.

From a biologist's point of view, sequence comparison alone cannot provide some required information. For instance, one cannot determine the similarity of two remotely homologous proteins by sequence comparison. Structural comparison must be performed in this case. The other applications of structural comparison and database searching include computing structure-based evolutionary distances, identifying protein functions, understanding functional mechanisms, finding binding sites or other functionally important regions of the proteins, etc.

Many methods have been proposed and implemented for structural comparison. Some of these methods are quite efficient for pair-wise comparison of structures. However, when performing a database search, all these methods practise exhaustive searching. They compare the query structure against the structures in the database one by one, and report the ones most similar to the query structure. So, when a query is searched against a very large database, these methods cannot provide fast response.

Our design philosophy is to adopt a filter-and-refine approach. We extract some important pieces of information from the structural database, pre-compute them, store them, and build an index on them. When evaluating a query, we use this index to prune away a large portion of the database. If needed, we only do the actual comparison or alignment against this small set of candidate structures. We refer to the proposed method as the ProtDex scheme. We conducted an experimental study on the ProtDex scheme, and our preliminary results show that ProtDex is up to 16 times faster than the popular DALI method for database searching task (without actual alignments), while its overall accuracy is only slightly inferior to that of DALI.

2. Problem Definition

This section discusses some of the basic concepts regarding the structures of proteins and definition of the tasks involved in structural comparison and structural database searching.

2.1. Protein Structures

A protein is composed of a sequence of *amino acid (AA)* residues. Certain portions of an AA sequence fold into particular shapes such as *helices* and *sheets* due to the forces of nature. Such a shape is called a *secondary structure element*

Proceedings of the Eighth International Conference on Database Systems for Advanced Applications (DASFAA'03)
0-7695-1895/03 $17.00 © 2003 IEEE

(SSE). The portions other than the SSEs have no specific shape, and are called *loop regions*. SSEs and loop regions combine together and fold again to form the 3-dimensional *polymer chains*. A protein molecule is composed of one or more polymer chains. (However, protein structure comparison and database searching tasks are usually performed at the level of polymer chains rather than at the level of entire proteins. Thus, when 'protein structure' is referred throughout the paper, it actually means the polymer chain structure of protein.)

The 3-dimensional structure of a protein is represented as a set of 3D positions (x, y, z co-ordinates) of all the atoms in a protein in a 3D reference frame. These co-ordinates are determined by the biologists using laboratory methods such as *nuclear magnetic resonance (NMR)* and *X-ray crystallography*.

The shape of a protein can be roughly determined by the positions of the central carbon atoms (C_α atoms) of the AA residues. Thus, a protein structure can be logically represented as a sequence of 3D positions of all the C_α atoms in the protein. When performing structural comparison, many methods take only C_α atoms into account, and simply neglect other atoms.

2.2. 3D Structural Comparison

Given two 3D protein structures, we need to compare them pair-wise according to a certain framework, and determine how similar they are. The framework for pair-wise structural comparison comprises i) representation of structures, ii) scoring schemes, iii) comparison algorithms, and iv) assessment of the results. The task of structural comparison is more difficult to define than that of sequence comparison. There may be several definitions for it, depending on the framework used. We will not discuss the various structural comparison frameworks in detail in this paper. Interested readers can refer to good survey papers like [4, 6].

Generally speaking, we try to superimpose one protein structure over another in order to obtain the minimum *root mean square deviation (RMSD)* between them. The smaller the RMSD, the more similar the structures. Superimposition involves translation and rotation of one structure to fit it onto the other so that the corresponding elements in two structures will be as close as possible. This process is also called *structural alignment*. However, the algorithm for finding the optimal alignment, in terms of minimal RMSD, between two structures having n and m residues respectively takes $O(n^4m^4)$ time [1]. It is impractical to be used on a large scale. In addition, more than one optimal alignment may also exist [6]. So, people use various approximation algorithms to solve the problem of structural alignment.

In order to assess the accuracy of a particular method, we need to compare the results against the *SCOP (Structural Classification of Proteins)* database [13], which is manually constructed, and used as a golden standard by the biologists [21]. If the two structures which are shown to be similar by a comparison method actually fall under the same group (class, fold, superfamily or family according to different accuracy requirements) in SCOP, this method is assumed to be accurate.

2.3. Structural Database Searching

Given a query protein structure, we need to search through the database and report the structures that are similar to the query structure. There may be a user-defined or pre-defined similarity threshold, and the structures whose similarity scores are equal to or above the threshold are reported. When using an exhaustive searching approach, searching through a database with N structures essentially means running pair-wise comparison N times. However, some fast but coarse searching methods such as [10] can be used as a pre-filter before performing the detailed database searching. In this way, the structures that are very unlikely to be included in the answer set can be pruned away after being glanced by a quick scan step before going into the expensive alignment step.

3. Related Works

As mentioned above, the problem of finding the optimal structural alignment or comparison is very computationally expensive. So, a number of approximation methods for nearly optimal solutions have been developed to solve this problem.

Some methods address only the problem of pair-wise comparison of two structures in general [5, 17, 18, 19]. But others explicitly address the problem of similarity searching in the structural databases in addition to pair-wise comparison [10, 11, 12, 15, 16]. Some methods use fine-grain approach, i.e., they first work at the SSE level, and then go into the AA-residue level for detailed processing [5, 11, 12, 18, 19]. But other methods use coarse-grain approach, and treat only SSEs as the basic elements [10, 15, 16]. Naturally, coarse-grain methods are faster but less accurate than the fine-grain ones. Some survey reports [4, 6, 21] are available for reviewing the background theories and the various methods in this area.

The main weakness of the existing methods is that when performing a database search, they compare the query structure against all the 3D structures (or their representative structures) in the database. Some methods use quick pre-filtering before performing the detailed alignment. Even in this case, due to the lack of pre-computed and stored data, each structure in the database has to be scanned at least once. Exhaustive searching can offer quite a satisfactory

response time until today since the number of structures in the databases is still of the order of thousands. However, given the rapid growth rates of the structural databases in the near future, such an exhaustive searching will be prohibitedly expensive to be performed.

4. The ProtDex Method

ProtDex stands for **Prot**ein In**Dex**ing. In this method, we use filter-and-refine approach. First, we build an inverted file index on the properties of the protein structures in the database. When evaluating a query, with the help of this index, we determine the overall or global similarity scores of all the protein structures in the database with respect to the query protein structure without doing any actual alignment. Then, we filter out and report a small number of top scoring candidate proteins to the user. After that, we can optionally execute the actual pairwise structural alignment of each candidate protein with the query protein. In filer-and-refine framework, our ProtDex method corresponds to the first filtering step. We can use any existing structural alignment algorithm to perform the second refinement step.

We use an intra-molecular distance matrix of C_α atoms to represent each protein structure. The design rationale behind our approach is the following: two protein structures are similar if their distance matrices are similar, and two distance matrices are similar when they are composed of similar (or exactly the same) sub-matrices. The more similar the two matrices, the greater the number of similar sub-matrix pairs they contain.

To facilitate efficient database searching, first we build the distance matrices for all the protein structures in the database. Then, we break the distance matrices into smaller fixed-size sub-matrices, and build an index on them. Later, this index is used to find the structures that are similar to the query structure. The approach we use is similar to the method used in the original version of DALI [12]. However, while DALI uses Monte Carlo optimization to assemble the sub-matrices to yield the actual alignment on the fly, we build an index on the sub-matrices and stores the index for future database searchings.

4.1. Extracting Information from PDB Database

PDB (Protein Data Bank) (http://www.pdb.org/) [2] is the most popular and widely used resource in structural genomics. It stores the structural information and annotations on several bio-molecules: mainly proteins, some nucleic acids and carbohydrates as well. Each bio-molecule is identified by a unique *PDB id* in the format $naaa$, where n is an integer, and a is an alphanumeric character (for example, *1tph*). A molecule may have more than one polymer chain. In such a case, each chain is referred to in the format *PDB id - chain id* (for example, *1tph-1*). For each bio-molecule, PDB stores the 3D co-ordinates of all the atoms that compose the molecule, alongside with other information (such as chain information, SSE information and annotations). PDB stores information on each bio-molecule as a separate text file. To build an index on the protein structures, we extract only the following information from the PDB files:

For each protein chain in PDB file:
PDB id - chain id; No. of AA residues; No. of SSEs
For each AA Residue:
3D coordinate (x, y, z) of C_α carbon
For each SSE:
SSE type (Helix or Sheet); SSE Start position; SSE length

Parsing PDB files is not a trivial task as they are quite complex and sometimes have irregularities in them [8]. In our implementation, we do not reinvent the wheel for PDB file parsing, but use the *Protein Parser* of *DaliLite* suite [9], which is the standalone version of DALI, for this purpose. DaliLite in turn uses *DSSP (Definition of Secondary Structure of Proteins)* program [14] to extract SSE information.

4.2. Building Intra-molecular Distance Matrices

Let us consider our imaginary protein $9xxx$ with 7 AA residues. When considering the structure of a protein, we only take C_α atoms (residue centers) into account, and simply ignore other atoms.

Let d_{ij} be the Euclidean distance between C_α of residue i and C_α of residue j (measured in Å). Clearly, $d_{ij} = 0$ when $i = j$. We can represent the structure of the protein $9xxx$ with an *intra-molecular distance matrix* (*distance matrix* for short) as shown in Figure 1. The distance matrix is symmetrical along the main diagonal. $d_{ij} = d_{ji}$.

	1	2	3	4	5	6	7
1	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}	d_{17}
2	d_{21}	d_{22}	d_{23}	d_{24}	d_{25}	d_{26}	d_{27}
3	d_{31}	d_{32}	d_{33}	d_{34}	d_{35}	d_{36}	d_{37}
4	d_{41}	d_{42}	d_{43}	d_{44}	d_{45}	d_{46}	d_{47}
5	d_{51}	d_{52}	d_{53}	d_{54}	d_{55}	d_{56}	d_{57}
6	d_{61}	d_{62}	d_{63}	d_{64}	d_{65}	d_{66}	d_{67}
7	d_{71}	d_{72}	d_{73}	d_{74}	d_{75}	d_{76}	d_{77}

	1	2	3	4	5	6	7
1	0	4	5	5	8	9	9
2	4	0	4	5	7	9	9
3	5	4	0	3	6	13	12
4	5	5	3	0	6	12	12
5	8	7	6	6	0	10	10
6	9	9	13	12	10	0	8
7	9	9	12	12	10	8	0

Figure 1. Distance matrix of $9xxx$ and its sample entries.

The advantage of such representation is that it is immune to the relative 3D position and rotation of the structure. The

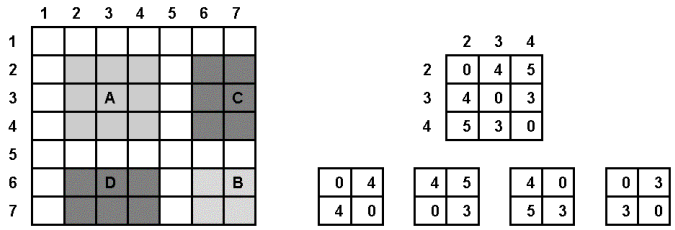


Figure 2. (a) Contact patterns in $9xxx$. (b) Splitting contact pattern A into fixed-size matrices

only information it cannot store is the overall chirality (left-handedness or right-handedness) of the structure [12]. So, using distance matrices, the expensive operations of translation and rotation need not be done for structural comparison. We only need to detect the similarity of the two entire distance matrices or certain portions of them.

In our example, let us assume that the residue number 2 to 4 form a helix SSE, and 6 to 7 form a sheet SSE. The intersection of two SSEs forms a *contact pattern*. In Figure 2(a), the contact patterns A and B represent the intra-structural configurations of each SSE respectively, and the contact patterns C and D represent the inter-structural configurations of the SSEs.

According to some observations such as [11, 12], in a protein, the shapes and placements of the SSEs mainly determine the overall structure of it. As an effect, when a protein is represented as a distance matrix, the majority of important information on the protein structure is conserved in the SSE contact patterns. So, it will be enough to perform the comparisons between the corresponding contact patterns of two proteins in order to detect their structural similarity. With a view to restrict the amount of information to be stored, we only take care of those C_α–C_α distances that are in the contact patterns, and simply neglect the others.

The contact patterns are symmetrical along the main diagonal. For example, pattern D is the transposed matrix of pattern C. So, it is sufficient to take only the contact patterns that are on or above the main diagonal into account. We can eliminate the contact pattern D in this case.

4.3. Cutting Fixed-size Matrices and Extracting Properties

Since the lengths of the SSEs are variable, the sizes of the contact pattern matrices also vary. It may not be easy to index these variable size matrices as they are. So, we cut these contact pattern matrices into $n \times n$ fixed-size overlapping sub-matrices. Let us take n = 2 for example. The contact pattern A can be cut into four 2×2 sub-matrices as shown in Figure 2(b). To build an index, we need to extract and store the useful information from each $n \times n$ sub-matrix. For an $n \times n$ matrix, we store:

1. its grand sum (sum of all $n \times n$ cells)
2. its contact pattern type (Helix-Helix = HH, Helix-Sheet = HE, or Sheet-Sheet = EE)
3. its contact pattern size (number of rows × number of columns)
4. its contact pattern ordinal number (ordinal number of row SSE, ordinal number of column SSE)
5. its relative starting position (row, column) within the contact pattern.

For instance, referring to Figure 1 and 2(a):

- For the 2×2 sub-matrix starting at the cell (2, 2), we store the values: **8, HH, (3×3), (1,1), (1,1)**
- For the 2×2 sub-matrix starting at the cell (3,6), we store the values: **49, HE, (3×2), (1,2), (2,1)**, etc.

In our actual implementation, we use 6×6 sub-matrices, because the minimum length of an SSE we use is 6.

4.4. Building Inverted File Index

Inverted file indexing has been successfully used in the area of text indexing [3] for a long time. Recently, it has also been successfully used for indexing the genomic sequence databases [20]. Our ProtDex method is the first attempt to use inverted file indexing for structural databases.

We build our inverted file index in such a way that each unique $n \times n$ fixed-size matrix points to a list of proteins in which it occurs. We do not store the $n \times n$ matrix directly. Instead, to approximate an $n \times n$ fixed-size matrix, we only store an information pair containing its grand sum (sum of all cells) and the type of contact pattern in which it occurs. Such a unique pair is called an *index term*. The entire collection of all these unique pairs is called the *index term list*. Each index term points to a list of the PDB ids in which the pair occurs, together with the size (number of rows × number of columns) and the ordinal number of the contact pattern in which it occurs, and its relative position (row, column) in the contact pattern. This list is called the *posting list*. Both index term list and posting lists are sorted for easier retrieval. A sample portion extracted from an inverted file index is shown in Figure 3.

There are many ways to organize the index term list and the inverted lists [3]. These methods include B^+-tree, linked lists, etc. In our implementation, simple sequential

Index Term [grand sum, contact pattern type]	Posting Lists [PDB id, contact pattern size, ordinal contact pattern no., relative position]
[1000, EE]	[101m,(16×16),(1,1),(8,8)], [12asA,(18×8),(2,2),(11,2)]
[1000, EH]	[12asA,(19×12),(2,3),(11,2)], [1cm7B,(20×9),(1,2),(10,2)] [1cm7B,(20×9),(1,2),(5,2)]
[1000, HH]	[1bho1,(8×8),(1,3),(2,2)]
[1001, EE]	[12asA,(8×6),((2,7),(2,2)], [3hfmY,(17×6),(1,1),(5, 1)]
…	…

Figure 3. A sample inverted file index for protein structure database.

sorted lists are used for both structures. The index term list is small enough to store in the main memory. The respective posting list for each index term can be quite large, and must be stored on the disk.

4.5. Searching a Protein Structure

When a query structure is submitted, it is also parsed in the same manner as described above. A small inverted file index is also built in the same way (but excluding *PdbId* from the posting list). Then we compare the two sorted lists of the query index terms and the database index terms. For each matching index term, we compare their posting lists pair-wisely.

The protein structures in the database are ranked according to their similarity to the query structure. The scoring function used is based on $\sum(Term\ Weight \times Inverse\ Document\ Frequency)$ framework normally used in text databases. However, we have to modify this basic scoring function to suit the similarity matching nature of our application. In order to compute scores for each protein structure in the database, we have to first compute the following values:

1. **Grand sum distance**

$$D_{GSum}(i,j) = |GrandSum_i - GrandSum_j| \quad (1)$$

where i is a query index term and j is a database index term.

2. **Contact pattern area deviation**

$$D_{Area}(a,b) = \frac{|A_a - A_b|}{A_a} \quad (2)$$

where

$$A_x = CPatternSize_x.NumRow \times CPatternSize_x.NumCol$$

$(x \in \{a,b\})$, where a is a query posting list entry and b is a database posting list entry.

3. **Contact pattern aspect ratio deviation**

$$D_{ARatio}(a,b) = \frac{|V_a - V_b|}{V_a} \quad (3)$$

where

$$V_x = min \begin{cases} CPatternSize_x.NumRow/CPatternSize_x.NumCol \\ CPatternSize_x.NumCol/CPatternSize_x.NumRow \end{cases}$$

$(x \in \{a,b\})$, where a is a query posting list entry and b is a database posting list entry.

4. **Contact pattern ordinal number deviation**

$$D_{Ordinal}(a,b) = \left| \frac{|OrdinalNumber_a.row - OrdinalNumber_a.col|}{numSSE_a} - \frac{|OrdinalNumber_a.row - OrdinalNumber_a.col|}{numSSE_b} \right| \quad (4)$$

where a is a query posting list entry and b is a database posting list entry. $numSSE_a$ is the number of SSEs in the query and $numSSE_b$ is the number of SSEs in the corresponding protein ($PdbId$) in the database.

5. **Fixed-size matrix relative position distance**

$$D_{Pos}(a,b) = L1(RelativePosition_a.(row,col), RelativePosition_b.(row,col)) \quad (5)$$

where a is a query posting list entry and b is a database posting list entry, and $L1$ is the Manhattan distance function.

6. **Fixed-size matrix count deviation**

$$D_{FMCount}(Q,P) = \frac{|numFixedMatrix_Q - numFixedMatrix_P|}{numFixedMatrix_Q} \quad (6)$$

where Q is the query protein and P is a database protein.

In the comparison process, we need to define the following thresholds:

Grand sum distance threshold	T_{GSum}
Contact pattern area deviation threshold	T_{Area}
Contact pattern aspect ratio deviation threshold	T_{ARatio}
Contact pattern ordinal number deviation threshold	$T_{Ordinal}$
Fixed-size matrix relative position distance threshold	T_{Pos}
Fixed-size matrix count deviation threshold	$T_{FMCount}$

The two index terms i and j are regarded as *matched* if and only if:

$$(CPatternType_i = CPatternType_j) \wedge (D_{GSum}(i,j) < T_{GSum})$$

The two posting list entries a and b are regarded as *matched* if and only if:

$$(D_{Area}(a,b) < T_{Area}) \wedge (D_{ARatio}(a,b) < T_{ARatio}) \wedge (D_{Ordinal}(a,b) < T_{Ordinal}) \wedge (D_{Pos}(a,b) < T_{Pos})$$

For a query Q and a particular protein P, the similarity score for the global similarity of the two whole structures $S(Q,P)$ is calculated as:

$$S(Q,P) = W_{FMCount}(Q,P) \times W_{GSum}(i,j) \times \sum_{(match(i,j))}(W_{Term}(i) \times \max_{(match(a,b) \wedge PdbId_b = P)}(W_{Area}(a,b) \times W_{ARatio}(a,b) \times W_{Ordinal}(a,b))) \quad (7)$$

where **Fixed-size matrix count weight**

$$W_{FMCount}(Q,P) = \begin{cases} \exp^{-\frac{D_{FMCount}(Q,P)}{T_{FMCount}}} & \text{if } D_{FMCount}(Q,P) < T_{FMCount} \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

The purpose of $W_{FMCount}$ is to compensate the effect that the large proteins being matched and scored more frequently than the small ones.

Other weights for each matching index term pair and matching posting list entry pair are calculated as:

Term weight

$$W_{Term}(i) = \lg(\frac{N}{f_i}) + 1 \quad (9)$$

where N is the number of proteins in the database and f_i is the number of proteins that contain the query index term i. The purpose of W_{Term} is to add more weight to the query index terms that rarely occur in the database.

Grand sum distance weight

$$W_{GSum}(i,j) = \frac{T_{GSum} - D_{GSum}(i,j)}{T_{GSum}} \quad (10)$$

Contact pattern area deviation weight

$$W_{Area}(a,b) = \frac{T_{Area} - D_{Area}(a,b)}{T_{Area}} \quad (11)$$

Contact pattern aspect ratio deviation weight

$$W_{ARatio}(a,b) = \frac{T_{ARatio} - D_{ARatio}(a,b)}{T_{ARatio}} \quad (12)$$

Contact pattern ordinal number deviation weight

$$W_{Ordinal}(a,b) = \frac{T_{Ordinal} - D_{Ordinal}(a,b)}{T_{Ordinal}} \quad (13)$$

We do not use the weight of relative position distance in the scoring scheme. In our experiments, we use the values $T_{GSum} = 10$, $T_{Area} = 0.37$, $T_{ARatio} = 0.2$, $T_{Ordinal} = 0.3$, $T_{Pos} = 7$, $T_{FMCount} = 1.1$.

The answer is ranked according to $S(Q,P)$ (Equation 7) and then reported. We can optionally define the cutoff score T_{Score} in order to produce only a report of proteins with the scores above or equal to T_{Score}. However, setting the cutoff score has only a negligible effect on the running time, as we calculate the scores of all the proteins in the database simultaneously. For example, reporting the top 10 scorers and the top 1,000 scorers virtually take the same amount of time.

Table 1. Query proteins.

PDB id	# AA residues	SCOP Class	SCOP Family	# structures in same family
1mbd	153	All-α	a.1.1.2	716
1tph-1	245	α/β	c.1.1.1	109
8fab-A	103	All-β	b.1.1.1	1061

5. Discussion

SCOP v1.59 database (released May 1, 2002) contains 39,893 domain entries representing 31,418 protein chains extracted from 15,979 PDB files. The database we use in our experiments is a subset of SCOP v1.59 and contains 28,655 protein chains from 14,677 PDB files. The reason for excluding some structures from our database is that those structures cannot be parsed by DaliLite Protein Parser, the tool we use to parse the PDB files.

In order to assess the efficiency and effectiveness of our ProtDex method, we conduct an experiment that is similar to the one conducted by Singh and Brutlag [21]. We choose 3 representative proteins from 3 different SCOP classes as queries as shown in Table 1.

5.1. Performance

We run the database searching tests for both *DaliLite* [9] (the standalone version of DALI) and ProtDex on the same machine (Sun Ultra Sparc II with two 480MHz CPUs and 4GB memory). We use the default settings for DaliLite. We use the database mentioned above as the target database for both methods. The time statistics of the three sample queries for both methods are shown in Table 2. The results show that ProtDex can be approximately up to 16 times faster than DaliLite.

Table 2. Performance comparison.

PDB id	DaliLite(hh:mm:ss)	ProtDex(hh:mm:ss)
1mbd	03:27:42	00:08:34
1tph-1	00:14:42	00:05:47
8fab-A	00:10:07	00:00:38

5.2. Accuracy

The results provided by DaliLite are not comprehensive enough to conduct an accuracy test. For example, when running a database search against 28,655 protein chains with *1mbd* as the query, DaliLite gives an answer set containing only 10 protein chains! To obtain the comprehensive

answers by DALI method, we submit the queries to DALI server (http://www.ebi.ac.uk/dali/) and request the results. (However, we do not use the running time statistics of DALI Server because the machine they use may be different from ours.)

At the time of query (as of September 2002), the database used by DALI server is one that was last updated on June 16, 2002. So, we can safely assume that all the PDB chains that are in the target database (which is a subset of SCOP v1.59 released on May 1, 2002) used for ProtDex are also present in DALI server database. From the results returned by DALI server, we prune away the PDB chains that are not present in our target database so as to ensure a fair comparison.

We compare both the results from DALI and ProtDex against *SCOP* classifications [13] (http://scop.mrc-lmb.cam.ac.uk/scop/) which is regarded as the golden standard by the biologists. SCOP classification hierarchy is made of 4 levels: *class*, *fold*, *superfamily* and *family* — among which family is the most detailed classification. In our experiment, if a protein in the result set falls into the same SCOP family as the query protein, it is counted as a true positive. The accuracy (sensitivity vs. coverage) comparisons for both methods are shown in Figure 4. 655 answers for the three proteins 1mbd, 1tph-1 and 8fab-A, respectively.

Sensitivity and coverage are calculated as follows.

$$Sensitivity = \frac{Number\ of\ true\ positives}{Total\ number\ retrieved} \times 100\% \quad (14)$$

$$Coverage = \frac{Number\ of\ ture\ positives}{Number\ in\ same\ family} \times 100\% \quad (15)$$

It is generally observed that the overall accuracy of DALI is better than that of ProtDex. In the case of *1tph-1*, DALI can maintain its 100% sensitivity level (i.e. without any false positives) up to 91% coverage on the same family. However, we can also notice the steep inclines in the graphs of DALI. This is because DALI does not report the answers with Z-scores less than 2. So, it will never be able to cover the remotely homologous proteins in the same family as the query protein.

The inferiority in accuracy of ProtDex is due to its inherent weakness in using approximations for the fixed-sized matrices. We approximate an $n \times n$ matrix only by its grand sum. In fact, two $n \times n$ matrices may have the same grand sum value although they are actually very different. Thus, a lot of false positives can be introduced in the process of index term matching. However, on the other hand, matching the grand sums alone, instead of matching two whole $n \times n$ matrices in cell-by-cell fashion, can greatly speed up the process. In other words, we sacrifice accuracy in some degree to gain speed.

5.3. Pros and Cons of ProtDex

The clear advantage of ProtDex over other methods is its speed. Since we do not need to scan through each structure in the database, but use the index instead to rank the structures according to their degree of similarity to the query structure, much CPU time and disk access time can be saved.

ProtDex also has its own disadvantage when compared to other methods. Although it can provide the candidate answer set of proteins with their respective rankings, it cannot provide the actual alignment of the query structure and the database structures. It cannot inform which AA residue or SSE in the query actually corresponds to which one in a database structure. So, it can only be used as a pre-filtering step when the actual alignment is also required.

The second disadvantage is the storage overhead for the index structure. However, the size of the index is small compared to the overall size of the structural database. In our experiment, while the size of the original database with 14,677 PDB files (including annotations) is 8.46GB, the size of the entire index structure is only 1.2GB.

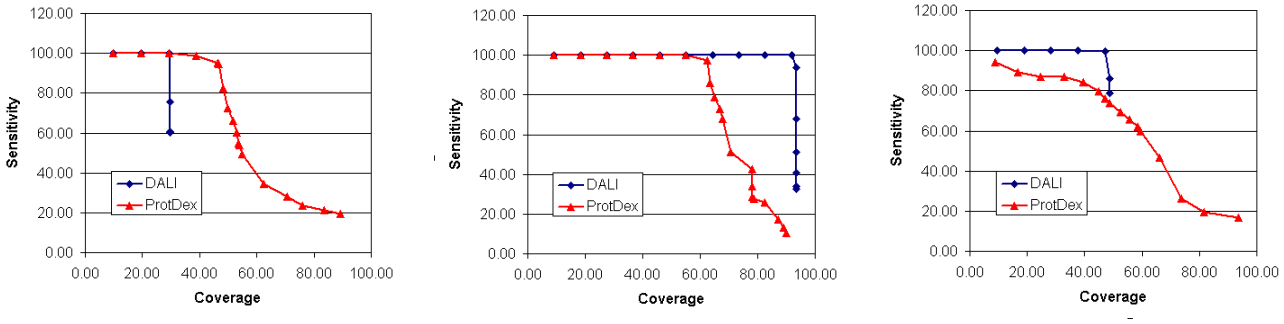
The third disadvantage is the time requirement to build and update the index. In our experiment, building the entire index on 28,655 protein chains (extracted from 14,677 PDB files) takes 30 minutes and 38 seconds. However, unlike the indexes used in many other applications, the indexes for protein structure databases may not require online updating. Although the database may be updated daily (with 10s of the new structures added per day), the updating of the index can only be done in batch on a regular basis (e.g. once per week) without much affecting the quality of service.

6. Conclusion

In this paper, we have proposed a new inverted file index-based approach for protein structure database searching. Preliminary experimental results show that this method is very efficient and reasonably effective. It can be perfectly used as a pre-filtering mechanism before going into a detailed database search by other more sensitive but slower structural alignment methods. Moreover, there are still rooms for improvements both in terms of efficiency and effectiveness. Since inverted file indexing is a well-established method in the area of text indexing and retrieval, we believe we can further improve the proposed scheme by borrowing many ideas from this area.

Acknowledgement

We thank the authors of *DaliLite Suite* [9] for kindly providing us with the software and helpful explanations on how it works.



(a) 1mbd (b) 1tph-1 (c) 9fab-A

Figure 4. Accuracy comparison.

References

[1] T. Akutsu, "Protein Structure Alignment Using a Graph Matching Technique", *Proceedings of Genome Informatics Workshop 1995*, Universal Academy Press, Tokyo, 1995, pp. 1–8.

[2] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne, "The Protein Data Bank", *Nucleic Acids Research*, 28, 2000, pp. 235–242.

[3] E. Bertino, B.C. Ooi, R. Sacks-Davis, K-L. Tan, J. Zobel, B. Shidlovsky, and B. Catania, *Indexing Technique for Advanced Database Systems*, Kluwer Academic Publishers, 1997.

[4] I. Eidhammer, I. Jonassen, and W.R. Taylor, "Protein structure comparison and structure patterns", *Journal of Computational Biology*, 7(5), 2000, pp. 685–716.

[5] J-F. Gibrat, T. Madej, and H. Bryant, "Surprising similarities in structure comparison", *Current Opinion in Structural Biology*, 6, 1996, pp. 377–385.

[6] A. Godzik, "The structural alignment between two proteins: is there a unique answer?", *Protein Science*, 5(7), 1996, pp. 1325–1338.

[7] J. Gorodkin, H.H. Strfeldt, O. Lund, and S. Brunak, "MatrixPlot: visualizing sequence constraints", *Bioinformatics*, 15, 1999, pp. 769–770.

[8] C.W.V. Hogue, "Structure databases", *Bioinformatics: a practical guide to the analysis of genes and proteins*, Wiley-Liss Inc., 2001, pp. 83–109.

[9] L. Holm and J. Park, "DaliLite workbench for protein structure comparison", *Bioinformatics*, 16(6), 2000, pp. 566–567.

[10] L. Holm and C. Sander, "3-D lookup: fast protein structure database searches at 90% reliability", *Proceedings of 3rd International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, 1995, pp. 179–187.

[11] L. Holm and C. Sander, "Mapping the protein universe", *Science*, 273, 1996, pp. 595–602.

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

[13] T.J.P. Hubbard, B. Ailey, S.E. Brenner, A.G. Murzin, and C. Chothia, "SCOP: a structural classification of proteins database", *Nucleic Acids Research*, 25(1), 1997, pp. 236–239.

[14] W. Kabsch and C. Sander, "DSSP: definition of secondary structure of proteins given a set of 3D coordinates", *Biopolymers*, 22, 1983, pp. 2577–2637.

[15] A. Martin, "The ups and downs of protein topology: rapid comparison of protein structure", *Protein Engineering*, 13, 2000, pp. 829–837.

[16] T. Ohkawa, S. Hirayama, and H. Nakamura, "A method of comparing protein structures based on matrix representation of secondary structure pairwise topology", *Proceedings of 4th IEEE Symposium on Intelligence in Neural and Biological Systems*, 1999, pp. 10–15.

[17] C.A. Orengo and W.R. Taylor, "SSAP: sequential structure alignment program for protein structure comparison", *Methods in Enzymology*, 266, 1996, pp. 617–635.

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

[19] A.P Singh and D.L. Brutlag, "Hierarchical protein structure superposition using both secondary structure and atomic representations", *Proceedings of 5th International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, 1997, pp. 284–293.

[20] H.. Williams and J. Zobel, "Indexing and retrieval for genomic databases", *IEEE Transactions on Knowledge and Data Engineering*, 14(1), 2002, pp. 63–78.

[21] http://cmgm.stanford.edu/ brutlag/Papers/singh00.pdf