ABSTRACT

Motivation: To investigate structure-function relationships, life sciences researchers usually retrieve and classify proteins with similar substructures into the same fold. A manually constructed database, SCOP, is believed to be highly accurate; however, it is labor intensive. Another known method, DALI, is also precise but computationally expensive. We have developed an efficient algorithm, namely, index-based protein substructure alignment (IPSA), for protein-fold classification. IPSA constructs a two-layer indexing tree to quickly retrieve similar substructures in proteins and suggests possible folds by aligning these substructures.

Results: Compared with known algorithms, such as DALI, CE, MultiProt and MAMMOTH, on a sample dataset of non-redundant proteins from SCOP v1.73, IPSA exhibits an efficiency improvement of 53.10, 16.87, 3.60 and 1.64 times speedup, respectively. Evaluated on three different datasets of non-redundant proteins from SCOP, average accuracy of IPSA is approximately equal to DALI and better than CE, MAMMOTH, MultiProt and SSM. With reliable accuracy and efficiency, this work will benefit the study of high-throughput protein structure-function relationships.

Availability: IPSA is publicly accessible at http://ProteinDBS.metmissouri.edu/IPSA.php.

Contact: ShyuC@mizzou.edu

Supplementary information: Supplementary data are available at Bioinformatics online.

1 INTRODUCTION

Protein tertiary structure classification has become an important research topic in computational and molecular biology. Since similar protein tertiary structures might have correlations with specific biological functions (Zarembinski et al., 1998), several structural genomics projects (Chen et al., 2004; von Goetzthuss et al., 2006) are focused on understanding the links between protein sequences, structures and functional properties. To understand protein structure-function relationships, life sciences researchers usually group proteins of similar substructures into a fold and study functional properties of these similar proteins. The computational methods of protein structure comparisons are usually focused on finding a detailed structural alignment between two proteins. The root mean square deviation (RMSD) is utilized to gauge the quality of alignment results from their optimized superimposition. Finding a global optimum of structural alignment is known to be an nondeterministic polynomial time (NP)-hard problem (Grodziki, 1996). To reduce the computational time, traditional structural alignment algorithms such as SSAP (Taylor and Orengo, 1989), DALI (Holm and Sander, 1993), CE (Shindyalov and Bourne, 1998), MAMMOTH (Ortiz et al., 2002) and MultiProt (Shatsky et al., 2004) apply heuristics and search locally optimal solutions.

Computational methods usually conduct one-against-all pairwise alignments between a newly discovered protein and all known proteins in the database to suggest protein folds. A known protein classification database, CATH (Pearl et al., 2003), is constructed using the SSAP algorithm. Another fold classification database, FSSP (Holm and Sander, 1994), is built based on the DALI algorithm. Since these classification works rely on heuristic-based structural alignments algorithms to measure the similarity of proteins, different heuristics may return divergent classification results for the same query structure. At present, a human curated database, SCOP (Murzin et al., 1995), is believed to maintain highly accurate classification results. Proteins with structural relationships are hierarchically grouped at the fold level. Even though manual inspection is more accurate, it is also labor intensive.

A consensus approach intersects classification results from multiple structural alignment algorithms to automate SCOP-fold classification (Can et al., 2004). With manually assigned weights for each individual method, this consensus approach yields an improved classification accuracy. However, a combination of structural alignment algorithms is known to be computationally expensive. With the advent of high-throughput techniques such as X-ray crystallography and nuclear magnetic resonance (NMR), the number of known protein structures has rapidly increased in recent years. To accelerate SCOP-fold classification, there is an urgent need for an efficient protein structure classification algorithm with satisfactory accuracy. Our previous work, ProteinDBS (Shyu et al., 2004), extracts 33 features from 2D distance matrices generated from protein 3D structures. ProteinDBS is able to quickly measure the global similarity of two proteins based on the Euclidean distance of the corresponding feature vectors. Extending the global similarities used in ProteinDBS, we developed an efficient protein classification tool, E-Predict (Chi et al., 2006), that assigns newly discovered proteins to possible SCOP folds in real time. Other global similarity measurements, such as SGM (Rogen and Fain, 2003), PCC (Zhou et al., 2006) and SSM (Kriisine1 and Henrick, 2003).
2 METHODS

Protein structure retrievals can be conceptually considered as an application in the field of information retrieval (IR). For an IR system, types, orders and locations of terms make up the basic semantics of a document. Analogous to these concepts of IR, types, orders and topological relationships of protein substructure units can be identified to assist human inspections of protein folds.

2.1 Protein substructure unit extraction

Our algorithm compares protein tertiary structures in terms of matching protein substructure units. In order to extract the protein substructure units for comparisons, SSes, Helix (H) and Sheet (E), are first identified from tertiary structures. In our implementation, the identification of SSes is conducted by sequentially matching protein amino acid residues with the H and E templates of Spatial ARangement of backbone Fragments (SARF; Alexandrov, 1996). As shown in Figure 1, our substructure unit, \( u_i \), consisting of \( 2(2 \times 1) \) amino acids, is extracted by sliding a window of 3 amino acids within one SS (SSC), and another window of 1 amino acid in another SS (SSS). The substructure unit, \( u_i = l^1 \cup l^2 \), is defined as a concatenation of an opening segment with 3 amino acids (\( l^1 \)) and a closing segment with 1 amino acid (\( l^2 \)), where \( i \) and \( j \) are the starting residues of these two segments and \( i < j \). For small proteins with less than two SSS assignments, our method slides windows within the entire protein. For those identified SSes with more than 1 amino acid residues, sliding a window of 3-mers by one residue at a time, produces a large amount of substructure units. To reduce the search space, the sliding window of 3-mers is shifted every three amino acids. Since SSes usually contain more than five amino acids (Carl and John, 1999), \( i \) set to 5 in order to cover most of the protein secondary structures. With the sliding window, our algorithm can efficiently find the structural features and identify two proteins with same sequence of SSE but different in the overall fold. A large number of protein substructure units can be extracted from database proteins. In order to efficiently search proteins with similar substructures, our algorithm first maps protein tertiary structure into a series of 1D terms by comparing substructure units with our predefined substructure representatives. The algorithm then indexes these terms of the entire database proteins for fast searching.

2.2 Protein substructure representative generation

A protein substructure representative is conceptually similar to a stemmed word, which addresses all variations of terms sharing the same root words in an IR system. Each substructure representative is selected to represent a group of structurally similar substructure units. To compute RMSD of two substructure units, our method uses \( (x, y, z) \) coordinates of the \( C\alpha \) atom to model each amino acid residue. The substructure unit with \( (2 \times 1) \) amino acids is therefore converted into a \( (2l + 3) \)-D vector. RMSD is then measured by the Kabsch procedure (Kabsch, 1976), which superimposes 6-D vectors of two substructure units and optimizes the rotation and translation matrices. Two substructure units within a preset RMSD threshold of \( \eta \) are considered to be similar. In our implementation, \( \eta \) is set to 3.0 Å. Our method utilizes database indexing tree to quickly identify all unique substructure representatives in the entire protein database. This global collection of substructure representatives is named global substructure representatives (GSRs). The concept of identifying GSRs is to iteratively grow an indexing tree in a top-down manner. The leaf nodes of the indexing tree maintain the current GSRs. Algorithm 1 shows the pseudocode of generating GSRs. Range_Search is a function to query a substructure unit \( u_i \), which is a 6-D vector, against the indexing tree and retrieve current GSRs that are structurally similar to \( u_i \). If \( u_i \) is structurally similar to any existing GSR with RMSD \( \leq \eta \), it is then considered as a duplicate substructure representative and ignored. Otherwise, \( u_i \) will be inserted into the tree as a GSR with an unique term identifier. The algorithm starts with an empty indexing tree. The inner for-loop between lines 2 and 6 iteratively checks whether each substructure unit is a unique GSR or not. We chose an M-tree (Ciaccia et al., 1997) to index GSRs due to its high scalability for a large number of vector data. Also, M-trees can efficiently conduct Range_Searches in a multi-dimensional vector space. Algorithm 1 is applied to parse all protein tertiary structures in the database, \( D \), and grow a global indexing tree, which serves as a dictionary of all substructure representatives.

2.3 Mapping protein substructure unit into terms

The major purpose of creating the global indexing tree is to translate protein tertiary structure units into a series of terms, which make fast substructure retrieval possible. As discussed previously, each protein, \( P \), can be decomposed into a sequential set of \( n_P \) substructure units,
Index-based protein substructure alignment

Fig. 2. Two-layer term mapping: local mapping is to map an individual protein into a series of LSRs that are indexed in a local indexing tree; global mapping is to map each LSR into term identifiers of similar GSRs.

$SP = \{u_1, u_2, \ldots, u_n\}$. One intuitive way of translating a protein tertiary structure into terms is to conduct a Range_Search against the global indexing tree for each unit, $u_i$. From the search result, the unique term identifiers of returned GSRs are then assigned to $u_i$. Iteratively, searching in the global indexing tree, $n_a$ substructure units of $P$ are mapped into a list of $n_b$ terms, $f: SP \rightarrow T = \{t_1, t_2, \ldots, t_{n_b}\}$, where $n_b \geq n_a$. Since a Range_Search usually needs to execute computationally expensive RMSD comparisons for thousands of GSRs, the efficiency of the term mapping process needs to be addressed.

To tackle the efficiency issue, we introduce a two-layer term mapping mechanism. The first layer constructs a local indexing tree in terms of applying lines 2–6 of Algorithm 1 for a single protein, $P$. The purpose of constructing a local indexing tree is to generate a set of local substructure representatives (LSRs) for $P$. Instead of directly querying thousands of substructure units against the global indexing tree in the second layer, the algorithm queries only hundreds of LSRs. We will refer Figure 2 to discuss examples of the term mapping processing in this section. The algorithm of the first layer maps a group of similar substructure units from $P$ to an LSR, which is then queried against the global indexing tree in the second layer by conducting a Range_Search. The term identifiers of GSRs in the search result are indirectly assigned to the group of similar protein substructure units via the LSR. For example, the table in the upper-right corner of Figure 2 shows that $u_1$, $u_3$ and $u_5$ of protein $P$ are similar to a local substructure representative, LSR1. The algorithm conducts a Range_Search to query LSR1 in the global indexing tree; the returned term identifiers of similar GSRs, 1, 9, and 5, are associated with the local representative, LSR1, and are assigned to each of substructure units $u_1$, $u_3$, and $u_5$. Using the same procedure, all substructure units in the protein $P$ are converted into a list of terms. Normally, the size of a local indexing tree is much less than the size of the global tree. Therefore, the efficiency of term mapping is greatly increased.

2.4 Query processing

Given a newly discovered protein chain, the query process applies similar two-layer term mapping procedures as discussed in Section 2.3. However, instead of conducting a Range_Search in the global indexing tree for each LSR, this query process searches only the nearest neighbor from the global
The procedure that iteratively finds the longest chain alignment is described. In our chain-to-chain alignment, or ‘syntactic characterization’ approaches. We construct a data structure, \( P \).-H.Chi et al. \( X \) Longer alignment lengths mean that larger common substructures potentially score function is defined in the following equation.

\[
\text{score}(X, Y) = \frac{N_{XY}}{|X| + |Y| - \text{RMSD}(X, Y)}
\]

(1)

Longer alignments mean that larger common substructures potentially exist in \( X \) and \( Y \). Adding the RMSD value with additional 1.0 to avoid the singular condition. In our current implementation, \( N_{XY} \) is highly weighted by taking a square in the first ratio of Equation 1. Since a longer protein may potentially result in a longer alignment than a small protein, our scoring function is normalized by \( N_{XY} \). Also, the structural variation of aligned amino acid residues is penalized based on the RMSD value. The algorithm ranks a subset of database proteins with matched terms based on the similarity scores.

2.5.2 Term-to-term alignment In reality, each term usually has multiple occurrences in both a query protein and a database protein. In order to match substructure units of one protein with units of the other protein, our algorithm performs substructure alignments with topological constraints using a variant of dynamic programming. Given a term \( t \), the algorithm first sequentially identifies all occurrences of \( t \) in \( X \), \( A_t^X = \{t_{i_1}^1, t_{i_2}^2, \ldots, t_{i_m}^m\} \), where \( t_{i_j} \) denotes the 3D coordinates of the \( j \)-th occurrence of \( t \) in \( X \). The algorithm then accesses the inverted-protein index to find a subset of database proteins that are associated with \( t \). Such a database protein \( Y \) can be represented by an ordered sequence of all occurrences of \( t \), \( A_t^Y = \{t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_n}^n\} \), where \( j_{i_j} \) is the 3D coordinates of the \( i \)-th occurrence of \( t \). Our method employs a customized dynamic programming technique for finding the longest substructure alignment (LSA) between \( A_t^X \) and \( A_t^Y \). The algorithm first creates two data structures: (i) an aligned coordinate set \( \forall \) that keeps one-to-one corresponding axes of aligned substructures between \( X \) and \( Y \) and (ii) an alignment length matrix \( 2m \times 2n \) with dummy columns and rows between two consecutive occurrence of \( t \) in both \( X \) and \( Y \) as shown in Supplementary Figure S1. The first row and column are initialized by filling zeros. There are two types of cells in the matrix, namely term–term and dummy cells. The term–term cell has co-occurrence of terms from both \( X \) and \( Y \), while the dummy cell has existence of either a dummy row (\( \cdot \)) or a dummy column (\( \cdot \)). The rationale of inserting dummy elements is to pass the best alignment results of previous elements for use by later elements. The ‘if statement’ at line 7 in Algorithm 2 distinguishes these two type of cells. Lines 8-14 deal with term–term cells for growing the alignment and lines 16-30 handle dummy cells for updating the currently best alignment result. Let \( \{l_{ij}\} \) be the alignment length for aligning \( \{t_{i_1}^1, t_{i_2}^2, \ldots, t_{i_m}^m\} \) and \( \{t_{j_1}^1, t_{j_2}^2, \ldots, t_{j_n}^n\} \), where \( i \leq m \) and \( j \leq n \). The 3D coordinates of aligned substructures are kept in \( \theta_{l_{ij}} \) with a corresponding RMSD value, which is measured from the aligned substructures from both proteins. From lines 8-14, \( \{l_{ij}\} \) is increased by one when the 3D coordinates of \( \{t_{i_1}^1, t_{i_2}^2, \ldots, t_{i_m}^m\} \) can be superimposed within an RMSD threshold, \( \gamma \). Otherwise, \( \{l_{ij}\} \) is assigned to 1. Supplementary Figure S1 shows an example to explain the alignment process. Since the first pair of substructures \( \{t_{i_1}^1, t_{i_2}^2\} \) in \( A_t^X \) and \( A_t^Y \) are able to be aligned within an RMSD threshold, \( \gamma \). Another alignment result, \( \{4,4\} = 2 \), is due to the fact that two protein

\[
\text{Algorithm 2: LSA}
\]

1. for \( r = 1 \) to \( 2e \) do
2. for \( j = 1 \) to \( 2e \) do
3. if \( \max(\{l_{ij}\}) < 0 \) then
4. \( \theta_{l_{ij}} = 0 \)
5. end if
6. else
7. if \( \{l_{ij}\} = 0 \) then
8. \( \theta_{l_{ij}} = 0 \)
9. end if
10. if RMSD(\(\{\theta_{l_{ij}}\}\)) < \(\{\gamma\}\) then
11. \( \theta_{l_{ij}} = \{\gamma\} \)
12. end if
13. end if
14. end if
15. \( \{l_{ij}\} = 1 \)
16. end if
17. end for
18. end for
19. end if
20. end if
21. end if
22. \( \{l_{ij}\} = 0 \)
23. end if
24. end if
25. end if
26. end if
27. \( \{l_{ij}\} = 0 \)
28. end if
29. end if
30. end if
31. end if
32. end if
33. \( \{l_{ij}\} = 2 \)
34. end for
35. end for
substructures, \( (\mathbf{r}_1^i, \mathbf{r}_2^i) \) and \( (\mathbf{r}_2^j, \mathbf{r}_1^j) \), can be superimposed within an RMSD threshold. When adding a new pair of substructures in the current alignment results in an RMSD that exceeds \( \gamma \), the algorithm keeps the new substructures and assigns 1 to the cell, such as \([6,6]\) in Supplementary Figure S1. From lines 16-30 of Algorithm 2, a dummy element \([i,j]\) propogates alignment results from the previous stage by taking the maximal values from two neighbors, \( [i-1,j] \) and \( [i,j-1] \) (\( \rightarrow \)), as depicted in cell \([4,5]\) of Supplementary Figure S1. If these two neighbors have the same alignment length, the one with a smaller RMSD will be chosen.

In the process of aligning \( \mathcal{A}^\mathcal{S} \) and \( \mathcal{A}^\mathcal{S'} \), we define a scoring function in Equation (2) to evaluate the quality of alignment \( [i,j] \) based on the alignment length \( \|I_j\| \) and an RMSD value. To provide better accuracy in protein-fold classification, the top 1 result of term-to-term alignment is utilized to determine a candidate set of rotation matrices and translation vectors, which are used for superimposing two proteins and computing the similarity score in the chain-to-chain alignment.

### 2.6 Further refinement

During the term mapping procedure (see Section 2.3), Range Search is utilized to find a structurally similar unit with RMSD \( \leq \eta \). However, during experiments, we found that some proteins with dissimilar structure are also mapped to the same term, which would increase response time of IPSA. To further improve the efficiency of IPSA, we introduce type, angle and distance of substructure unit as new criteria into Range Search. The refined IPSA algorithm is called IPSA*.

Given a substructure unit \( u \) consisted of SSEs and SSEs, its type, denoted as \( T_u \), is a concatenation of type of SSEs and SSE (Helix or Sheet). To compute the angle and distance of substructure unit, we adopt the methods from Singh and Brodtk (1997). First, each SSE in the substructure unit is represented by its start point \( \Delta u_{\text{start}} \) and end point \( \Delta u_{\text{end}} \). Then, unit direction vector of each SSE is calculated with the start and end points as \( v = \Delta u_{\text{end}} - \Delta u_{\text{start}} \). For substructure unit \( u \), its angle \( \Psi \) is calculated by taking the inverse cosine of the dot product of the two unit direction vectors. The distance \( d \) is set to minimum value of the distances between corresponding start and end points of two SSEs.

With these new features, two substructure units \( u \) and \( u' \) are considered as similar if the following four conditions are met: (i) \( T_u = T_{u'} \), (ii) \( \Psi = \Psi' \leq \epsilon \), (iii) \( \|I_u - I_{u'}\| < d \) and (iv) RMSD \( \leq \eta \). Here \( \epsilon \) and \( d \) are thresholds for the angle and distance, respectively. It is noted that IPSA* is only used to process querying which requires quick response but low accuracy.

### 3 COMPUTATIONAL RESULTS

In this section, we investigate both the efficiency and accuracy for SCOP-fold classification and protein structure retrieval. From an evaluation work (Novotny et al., 2004), both DALI and CE are considered as accurate protein-fold comparison methods. Our proposed IPSA algorithms are compared with these two well-recognized algorithms and other representative algorithms, which include MAMMOTH (Ortiz et al., 2002), MultiProt (Shatsky et al., 2004) and SSM (Krivine and Henrick, 2004), using Non-Redundant Protein Data. The consensus approach (Can et al., 2004) is not compared since the weight for each individual alignment algorithm is data dependent.

#### 3.1 Non-redundant protein data

With proteins from the Protein Data Bank (PDB) (Berman et al., 2000), SCOP manually classifies structurally similar proteins into folds. Since structural similarity could be captured using sequence alignment tools, classifying redundant proteins which have high sequence similarities is basically considered to be a trivial case of fold classification. We therefore use non-redundant protein data for performance evaluation. The non-redundant dataset is selected using the latest version of PDBeSelect (Hobohm and Sander, 1994) with <25% sequence similarity. Small proteins with less than 20 amino acids are also excluded.

Our dataset is shown in Table 1, which is consisted of two parts: (i) test proteins for querying SCOP fold and (ii) database proteins. Given two consecutive SCOP releases \( v_1 \) and \( v_2 \) (\( v_1 \leq v_2 \)), \( \Delta v_{12} \) denotes a set of newly discovered proteins in \( v_2 \) that have not been identified in \( v_1 \). To mimic the SCOP-fold classification, we use the newly discovered proteins to construct the test proteins, which include at least one non-redundant and representative protein from each superfamily existing in \( \Delta v_{12} \). Similarly, the database proteins are constructed to ensure at least one non-redundant and representative protein from each superfamily in the entire SCOP space.

| Table 1. The number of non-redundant proteins in a test set of general and non-redundant test sets in \( \Delta v_{12} \) known.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets</td>
<td>Test proteins</td>
<td>Database proteins</td>
</tr>
<tr>
<td>I</td>
<td>640 proteins from ( \Delta v_{12} )</td>
<td>2767 proteins from SCOP ( v_1 )</td>
</tr>
<tr>
<td>II</td>
<td>607 proteins from ( \Delta v_{12} )</td>
<td>3015 proteins from SCOP ( v_1 )</td>
</tr>
<tr>
<td>III</td>
<td>610 proteins from ( \Delta v_{12} )</td>
<td>3276 proteins from SCOP ( v_1 )</td>
</tr>
</tbody>
</table>

| Table 2. The average response time of IPSA*, MAMMOTH, IPSA, MultiProt, CE and DALI for SCOP-fold classification per query
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Average response time (s)</td>
<td>Speedup of IPSA*</td>
</tr>
<tr>
<td>IPSA*</td>
<td>182</td>
<td>--</td>
</tr>
<tr>
<td>MAMMOTH</td>
<td>280</td>
<td>1.64</td>
</tr>
<tr>
<td>IPSA</td>
<td>402</td>
<td>2.21</td>
</tr>
<tr>
<td>MultiProt</td>
<td>655</td>
<td>3.60</td>
</tr>
<tr>
<td>CE</td>
<td>3071</td>
<td>16.87</td>
</tr>
<tr>
<td>DALI</td>
<td>9665</td>
<td>53.10</td>
</tr>
</tbody>
</table>

Due to the time complexity of the DALI algorithm, we evaluate the efficiency of SCOP-fold classification and structure retrieval using sampled non-redundant test proteins on a single server, which are selected from the protein dataset (III) listed in Table 1 based on distribution of number of amino acids in each protein. The list of the sampled proteins are shown in Supplementary Table S1.

We perform 3-fold experiments to compare the efficiency of IPSA with other algorithms, each using 50 proteins randomly selected from set (III). We measure the average response time of total 150 proteins to evaluate the efficiency of SCOP-fold classification and structure retrieval. The experiments are conducted on a Linux Fedora server with AMD Opteron dual-core 1000 series processors and 2 GB RAM. Table 2 shows that the fastest algorithm is IPSA*, which has 1.64, 16.87 and 53.10 times faster than MAMMOTH, CE and DALI, respectively.

2563
Table 1. of test proteins’ in Equation (3).

respectively. All the failed cases are excluded from the ‘total number
106 and 133 failed test cases for the protein sets (I), (II) and (III),
have such function. In addition, when running SSM, we have 82,
alignment to rank the alignment results of MultiProt as it does not
association between the accuracy and resolution.

2.5.2 SCOP-fold retrieval

proteins are
fold retrieval. With the retrieval results, the retrieved database
3.5 Structural alignment

In our experiment, the test protein is classified into the same
fold as the top-ranked database protein. To evaluate the accuracy
of SCOP-fold classification, we use a general metric, correct
classification rate (CCR), which is defined as follows:

\[
\text{CCR} = \frac{\text{The number of correctly classified proteins}}{\text{The total number of test proteins}}
\] (3)

Figures 3a presents the CCR performance comparison among IPSA,
IPSA*, DALI, CE, MAMMOTH, MultiProt and SSM using the
protein sets (I), (II) and (III) listed in Table 1. Intuitively, the optimal accuracy of SCOP-fold classification is 100% CCR. Our
classification results for the three sets show that IPSA exhibits
85.31%, 87.31% and 89.65% CCR. These accuracies are comparable
with those of DALI: 85.58%, 88.47% and 88.24% CCR. Also, IPSA
outperforms MAMMOTH and CE in both protein datasets for the
SCOP-fold classification by at least 3.28% and 7.74%, respectively.

As IPSA utilizes the substructure from PDB to classify
SCOP fold, we also investigate relationship between the accuracy
and quality of compared structures by randomly selecting 50-folds
from set (III) and calculating average CCR and resolution of PDB in
each fold. The results are shown in Supplementary Figure S2. The
correlation coefficient \( r = 0.06 \), which means no significant linear
association between the accuracy and resolution.

It is noted that we use the scoring function of IPSA chain-to-chain
alignment to rank the alignment results of MultiProt as it does not
have such function. In addition, when running SSM, we have 82,
106 and 133 failed test cases for the protein sets (I), (II) and (III),
respectively. All the failed cases are excluded from the ‘total number
of test proteins’ in Equation (3).

3.4 Accuracy—SCOP-fold retrieval

Our experiment utilizes the precision-recall (PR) curves and
F-measure (van Rijsbergen, 1979) to gauge the accuracy of SCOP-
fold retrieval. With the retrieval results, the retrieved database
proteins are relevant when the SCOP-fold labels of these proteins
match the fold label of a query protein. Otherwise, these proteins
are irrelevant. If there are \( k \) database proteins with the same fold
as a query protein, an ideal retrieval should rank these proteins
in the top \( k \) results. Precision and Recall, which have been
discussed previously, are two standard performance measurements
for evaluating an IR system. The average PR curves of protein sets
(I), (II) and (III) are shown in Supplementary Figure S3. From the
figure, we can see that IPSA and IPSA* have higher precision at all
recall levels.

Given a query protein, \( q \), the \( F \)-measure shown in Equation (4)
considers both Precision and Recall for the \( i \)-th relevant protein.

\[
F(i, q) = \frac{2 \times \text{Precision}(i) \times \text{Recall}(i)}{\text{Precision}(i) + \text{Recall}(i)}
\] (4)

Since there may exist more than one relevant protein for a query
protein, \( q \), we define a normalized, single measurement, \( F_{\text{Normalized}}(\text{Score}(q)) \)
in Equation (5), which takes the average of the sum of each
individual \( F \)-measure and normalizes this average sum into a value
between 0 and 1. When \( q \) is a relevant protein, a high \( F_{\text{Normalized}}(\text{Score}(q)) \) is expected.

\[
F_{\text{Normalized}}(\text{Score}(q)) = \frac{\sum_{i=1}^{n} F(i, q)}{2 \times \sum_{i=1}^{n} \text{Precision}(i) + \text{Recall}(i)}
\] (5)

Figures 3b shows the plots of \( F_{\text{Normalized}}(\text{Score}(q)) \) for IPSA, IPSA*,
DALI, CE, MAMMOTH and MultiProt using the protein sets (I),
(II) and (III) listed in Table 1. SSM is excluded as it only outputs
the top 1 result. From the results, DALI presents the best retrieval
accuracies, 72.59%, 70.97% and 72.84%, while IPSA exhibits
competitive retrieval accuracies, 70.14%, 68.40% and 71.57%. In
addition, IPSA outperforms MAMMOTH and CE in both protein
datasets with retrieval accuracies that are better by at least 11.33%
and 13.10%, respectively.

Fig. 3. The plots of (a) CCR and (b) \( F_{\text{Normalized}}(\text{Score}(q)) \) for IPSA, IPSA*, DALI, MAMMOTH, CE, MultiProt and SSM using the protein sets (I), (II) and (III) in Table 1.
alignment as reported by others, but IPSA is able to find a solution close to DALI.

4 DISCUSSIONS
In this article, we introduce the IPSA algorithm to efficiently compare protein structure and retrieve proteins with similar substructures. The algorithm first builds a two-layer indexing tree to convert protein substructures into terms, and then aligns these terms and chains to ensure one-to-one correspondence of amino acids between the two overlapped proteins. The experiment results show IPSA achieves speedups of over an order of magnitude and approximately equal accuracy compared with DALI.

One of the key applications for a large-scale method of protein structure comparison is the molecular evolution of protein repertoire. To understand the full picture of structural relationship between proteins, it would be desirable to perform all-against-all structural comparison of currently solved protein structures in an organism. Specifically, one would be interested to determine locally conserved substructures for each pair of proteins. According to the UniPrekIB/Swiss-Prot Human Proteome Initiative, there are 25 686 unique protein structures currently available. This corresponds to about 338 million of protein–protein structural comparisons. Using a current method for local structural alignment such as DALI and estimating the running time for a single protein-protein structure comparison to be 3 s, the whole task would require running for 112 days on a 100-node CPU cluster. The efficiency of the proposed algorithm allows to reduce the running time to 2 days making this task feasible in the nearest future.

ACKNOWLEDGEMENTS
The authors would like to acknowledge the constructive comments and suggestions from anonymous reviewers. The authors would also like to thank Dr Dong Xu of University of Missouri-Columbia for helpful discussions. We would also like to thank the program authors of DALI, CE, MAMMOTH, MultiProt and SSM who provided open-access tools for research use.

Funding: University of Missouri-Columbia Research Council (in part); the Shumaker Endowment in Bioinformatics.

Conflict of Interest: none declared.

REFERENCES