YAPS: Yet Another Protein Similarity

Tomáš Novosád, Václav Snášel*
VŠB - Technical University of Ostrava
Ostrava, The Czech Republic
Email: {tomas.novosad,vaclav.snasel}@vsb.cz

Ajith Abraham
*Machine Intelligence Research Labs (MIR Labs)
http://www.mirlabs.org Email: ajith.abraham@ieee.org

Jack Y Yang
Harvard University, PO Box 400888, Cambridge, Massachusetts 02140-0888, USA
Email: dr.jack.yang@gmail.com

Abstract

This article presents a novel method for measuring protein similarity based on their tertiary structure. The new method deals with suffix trees and classical information retrieval tasks, such as the vector space model, using tf-idf term weighing schema or using various types of similarity measures. Our goal to use the whole PDB database of known proteins, not just some kinds of selections, which have been studied in other works. For verification of our algorithm we are using comparisons with the SCOP database, which is maintained primarily by humans. The next goal is to be able to categorize proteins not included in the latest version of the SCOP database with nearly 100% accuracy.

1 Introduction

Analyzing three dimensional protein structures is a very important task in molecular biology. Most solutions are make use of the state-of-the-art technologies such as nuclear magnetic resonance (NMR) spectroscopy techniques or X-Ray crystallography as seen in the increasing number of PDB [16] entries: 56366 as of March 10, 2009. It was proved that structurally similar proteins tend to have similar functions even if their amino acid sequences are not similar to each other. Thus it is very important to find proteins with similar structures (even in part) from the growing database to analyze protein functions. Yang et al. [24] exploited machine learning techniques including variants of Self-Organizing Global Ranking, a decision tree, and a support vector machine algorithms to predict the tertiary structure of transmembrane proteins. Hecker et al. [10] developed a state of the art protein disorder predictor and tested it on a large protein disorder dataset created from Protein Data Bank. The relationship of sensitivity and specificity is also evaluated. Habib et al. [8] presented a new SVM based approach to predict the subcellular locations based on amino acid and amino acid pair composition. More protein features can be taken into consideration and consequently improves the accuracy significantly. Wang et al. [22] discussed an empirical approach to specify the localization of protein binding regions utilizing information including the distribution pattern of the detected RNA fragments and the sequence specificity of RNase digestion.

We present a novel method for analyzing three dimensional protein structure using suffix trees and classical information retrieval methods and schemes. Several studies were developed for indexing protein tertiary structure [5, 20]. These studies are targeted mainly at some kind of selection of the PDB database. The goal of this work is that we are taking into account the whole current PDB database and calculating the similarities of each protein in comparison to each other protein. The suffix tree is a very useful data structure which can discover common substructures of proteins in a reasonable time (linear or logarithmic time), depending on the implementation of the construction algorithm.

When the generalized suffix tree is constructed for all proteins appearing in the entire PDB database, we

are using similar methods, which were previously studied [26, 9, 3, 13] for measuring the similarity of proteins based on their three dimensional structure definition. Our work arises from the relations of amino acid residues defined by its dihedral angles rather then the relations between just the Alpha Carbon atoms. The relations between alpha carbons use DALI for example, when computing the distance matrix between alpha carbon atoms of a given protein. In the final stage we are building a vector space model which is very suitable for various information retrieval tasks and can be used for future studies of proteins relations.

Rest of the paper is organized as follows. In the background Section 2 we briefly describe the proteins, vector space model and suffix trees. In Section 3 the description of the conversion of three dimensional protein backbone structures into the data which can be indexed by suffix trees is provided. Section 4 is the characterization of the proposed algorithm for measuring the similarity of proteins using the vector space model. Finally in Section 5 , we provide the evaluation of the proposed algorithm in comparison with the SCOP database.

2 Background

2.1 Protein and Its Structure

Proteins are large molecules. In many cases only a small part of the structure - ***an active site*** - is directly functional, the rest existing only to create and fix the spatial relationship among the active site residues [11].

Chemically, protein molecules are long polymers typically containing several thousand atoms, composed of a uniform repetitive backbone (or mainchain) with a particular sidechain attached to each residue. The amino acid sequence of a protein records the succession of sidechains.

The polypeptide chain folds into a curve in space; the course of the chain defining a **folding pattern**. Proteins show a great variety of folding patterns. Underlying these are a number of common structural features. These include the recurrence of explicit structural paradigms - for example, $\alpha - helices$ and $\beta - sheets$ and common principles or features such as the dense packing of the atoms in protein interiors. Folding may be thought of as a kind of intramolecular condensation or crystallization [11].

2.1.1 Protein Databank - PDB

The PDB archive contains information about experimentally-determined structures of proteins, nucleic acids, and complex assemblies. As a member of the wwwPDB, the RCSB PDB curates and annotates PDB data according to agreed upon standards [16].

2.1.2 Dihedral angles

Any plane can be defined by two non-collinear vectors lying in that plane; taking their cross product and normalizing yields the normal unit vector to the plane. Thus, a dihedral angle can be defined by four, pairwise non-collinear vectors.

The backbone dihedral angles of proteins are called ϕ (phi, involving the backbone atoms C-N-C_α-C), ψ (psi, involving the backbone atoms N-C_α-C-N) and ω (omega, involving the backbone atoms C_α-C-N-C_α). Thus, ϕ controls the C-C distance, ψ controls the N-N distance and ω controls the C_α-C_α distance.

The planarity of the peptide bond usually restricts ω to be 180° (the typical trans case) or 0° (the rare cis case). The ϕ and ψ dihedral angles tend to be from -180° to 180°.

2.2 Vector Space Model

The vector model [1] of documents is dated back to 70th of the 20th century. In vector model there are documents and users queries represented by vectors.

We use m different terms $t_1 \dots t_m$ for indexing N documents. Then each document d_i is represented by a vector:

$$d_i = (w_{i1}, w_{i2}, \dots, w_{im}),$$

where w_{ij} is the weight of the term t_j in the document d_i.
An index file of the vector model is represented by matrix:

$$D = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{Nm} \end{pmatrix},$$

where i-th row matches i-th document, and j-th column matches j-th term.

The similarity of two documents is given by following formula:

$$sim(d_i, d_j) = \frac{\sum_{k=1}^{m} (w_{ik} w_{jk})}{\sqrt{\sum_{k=1}^{m} (w_{ik})^2 \sum_{k=1}^{m} (w_{jk})^2}}$$

For more information see [12, 15, 1].

2.3 Suffix Trees

A suffix tree is a data structure that admits efficient string matching and querying. Suffix trees have been studied and used extensively, and have been applied to fundamental string problems such as finding the longest repeated substring [23], strings comparisons [4], and text compression [17]. Following this, we describe the suffix tree data structure - its definition, construction algorithms and main characteristics.

2.3.1 Definitions

The following description of the suffix tree was taken from Dan Gusfield's book *Algorithms on Strings, Trees and Sequences* [7]. One major difference is that we treat documents as sequences of words, not characters. A suffix tree of a string is simply a compact trie of all the suffixes of that string. In more precise terms [25] Citation:

Definition 2.1. A suffix tree T for an m-word string S is a rooted directed tree with exactly m leaves numbered 1 to m. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty sub-string of words of S. No two edges out of a node can have edge labels beginning with the same word. The key feature of the suffix tree is that for any leaf i, the concatenation of the edge labels on the path from the root to leaf i exactly spells out the suffix of S that starts at position i, that is it spells out $S[i \dots m]$.

In cases where one suffix of S matches a prefix of another suffix of S then no suffix tree obeying the above definition is possible since the path for the first suffix would not end at a leaf. To avoid this, we assume the last word of S does not appear anywhere else in the string. This prevents any suffix from being a prefix to another suffix. To achieve this we can add a terminating character, which is not in the language that S is taken from, to the end of S

The suffix tree of the string *"I know you know I know you#"* (Figure 1). There are seven leaves in this example, marked as rectangles and numbered from 1 to 7. The terminating characters are also shown in this diagram.

In a similar manner, a suffix tree of a set of strings, called a generalized suffix tree [7], is a compact trie of all the suffixes of all the strings in the set [25]:

Definition 2.2. A generalized suffix tree T for a set S of n strings S_n, each of length m_n, is a rooted directed tree with exactly $\sum m_n$ leaves marked by a two number tuple (k, l) where k ranges from 1 to n and l ranges from 1 to m_k. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty sub-string of words of a string in S. No two edges out of a node can have edge labels beginning with the same word. For any leaf (i, j), the concatenation of the edge labels on the path from the root to leaf (i, j) exactly spells out the suffix of S_i that starts at position j, that is it spells out $S_i[j \dots m_i]$.

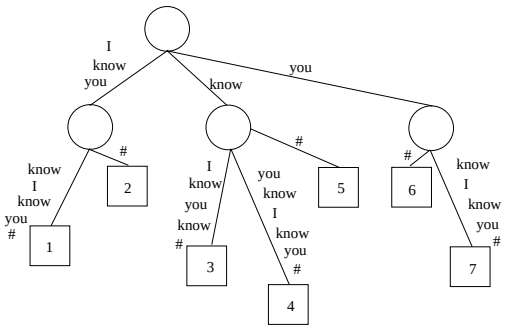

Figure 1. Simple example of suffix tree

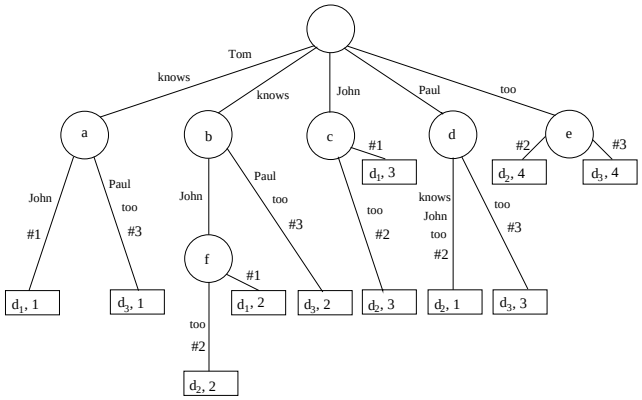

Figure 2. Example of the generalized suffix tree

Figure 2 is an example of the generalized suffix tree of the set of three strings - *"Tom knows John #1"*, *"Paul knows John too #2"* and *"Tom knows Paul too #3"* (#1, #2, #3 are unique terminating symbols). The internal nodes of the suffix tree are drawn as circles, and are labeled from a to f for further reference. Leaves are drawn as rectangles. The first number $(d_1, \dots, d_n)$, for later usage) in each rectangle indicates the string from which that suffix originates. The second number represents the position in that string where the suffix begins. Each string is considered to have a unique terminating character, which is not shown in this diagram.

2.3.2 Suffix Tree Construction Algorithms

The naive, straightforward method to build a suffix tree for a string S of length L takes $O(L^2)$ time. The naive method first enters a single edge for the suffix $S[1 \dots L]$ into the tree. Then it successively enters the suffix $S[i \dots L]$ into the growing tree for i increasing from 2 to L. The details of this construction method are not within the bounds of this article. The best references for constructing suffix trees in many different ways can be found on the Internet [6], or in the appropriate literature (e.g. [7] a good book on suffix tree construction algorithms in general).

Several linear time algorithms for constructing suffix trees exist [14, 21, 23]. To be precise, these algorithms also exhibit a time dependency on the size of the vocabulary (or the alphabet when dealing with character based trees): they actually have a time bound of $O(L \times min(\log |V|, \log L))$, where L is the length of the string and $|V|$ is the size of the language. These methods are more difficult to implement then the simple method, which is sufficiently suitable for our purpose.

We have also made some implementation improvements of the naive method to achieve better than the $O(L^2)$ worst-case time bound. With these improvements we have achieved constant access time when finding an appropriate child of the root and logarithmic time to find an existing child or to insert a new child node to any other internal nodes of the tree [13]. Next we have also improved the generalized suffix tree data structure to be suitable for large document collections [13].

3 Preparing the Data

We describe the process of retrieving the data for protein indexing. We used the whole PDB database which consists of approximately 49000 known proteins.

3.1 Creating Proteins Collection

In the current PDB database we can find proteins, nucleic acids and complex assemblies. Our study is focused just on relations between proteins. We have filtered out all nucleic acids and complex assemblies from the entire PDB database. Next we have filtered out proteins which have incomplete N-Cα-C-O backbones (e.g. some of the files have C atoms in the protein backbone missing, etc.). After this cleaning step we have obtained a collection consisting of 44351 files. Each file contains a description of one protein and its three dimensional structure and contains only amino acid residues with complete a N-Cα-C-O atom sequence. From each file we have retrieved at least one main chain (some proteins have more than one main chain) of at least one model (in some cases PDB files contains more models of three dimensional protein structure). In cases when the PDB file contains more main chains or more models we take into account all main chains of all models.

3.2 Encoding the 3D Protein Main Chain Structure for Indexing

To be able to index proteins by IR techniques we need to encode the 3D structure of the protein backbone into some sequence of characters, words or integers (as in our case). Since the protein backbone is the sequence of the amino acid residues (in 3D space) we are able to encode this backbone into the sequence of integers in the following manner. For simple example let's say the protein backbone consists of four amino acid residues $M\ V\ L\ S$ (abbreviations for methionie, valine, leucine and serine). The relationship between the two following residues can be described by its dihedral angles ϕ, ψ and ω. Since ϕ and ψ are taking values from the interval $\langle -180°, 180° \rangle$ we have to do some normalization. From this interval we have obtained 36 values (the interval was divided into 35 equal parts, by 10° degrees) e.g. $-180°$, $-170°$,...,0°,10°,...,180°. Each of these values was labeled with a positive number (00, 01, 02, ..., 35). Now, let's say that ϕ is $-21°$, the closest discrete value is $-20°$ which has the label 02, so we have encoded this dihedral with the string '02'. The same holds for ψ. The ω was encoded as the two characters A or B since the ω tends to be almost in every case 0° or 180°. After concatenation of these three parts we get a string which looks something like this 'A0102' which means that $\omega \approx 180°$, $\phi \approx -10°$, $\psi \approx -20°$

3.3 Summary

The major objective of this stage is to prepare the data for indexing by suffix trees. The suffix tree can index sequences. The resulting sequence in our case is a sequence of integers (positive numbers). For simple example let's say we have a protein with a backbone consisting of 6 residues e.g. M V L S E G with its three dimensional properties. The resulting encoded sequence can be for example:
{A3202, A2401, A2603, A2401, A2422}
After obtaining this sequence of 5 words, we create a dictionary of these words (each unique word receives its own unique integer identifier). The translated sequence will look like this:

{0, 1, 2, 1, 3}
In this way we encode each main chain of each model contained into one PDB file. This task is done for every protein included in our filtered PDB collection. Now we are ready for indexing proteins using suffix trees.

4 Protein Similarity Algorithm

We describe the algorithm for measuring protein similarity based on their tertiary structure. A brief description of the algorithm is detailed below:

1. Prepare the data as was mentioned in section 3.
2. Insert all encoded main chains of all proteins in the collection into the generalized suffix tree data structure.
3. Find all maximal substructures clusters in the suffix tree.
4. Construct a vector model of all proteins in our collection.
5. Build proteins similarity matrix.
6. For each protein find top N similar proteins.

4.1 Inserting All Main Chains into the Suffix Tree

In this stage of the algorithm we construct a generalized suffix tree of all encoded main chains. As mentioned in Section 3, we obtained the encoded forms of three dimensional protein main chains - sequences of positive numbers. All of these sequences are inserted into the generalized suffix tree data structure (section 2.3).

4.2 Finding All Maximal Substructure Clusters

To be able to build a vector model of proteins we have to find all maximal phrase clusters. As evident in 2, the phrases can be e.g. *"Tom knows John #1"*, *"knows John #1"*, *"John #1"*, etc. The **phrase** in our context is an encoded protein main chain or any of its parts. The document in our context can be seen as a set of encoded main chains of the protein. Now we can define a maximal phrase cluster (the longest common substructure) [26]:

Definition 4.1. A phrase cluster is a phrase that is shared by at least two documents, and the group of documents that contain the phrase. A maximal phrase cluster is a phrase cluster whose phrase cannot be extended by any word in the language without changing (reducing) the group of documents that contain it. Maximal phrase clusters are those we are interested in.

Now we simply traverse the generalized suffix tree and identify all maximal phrase clusters (i.e. all of the longest common substructures).

4.3 Building a Vector Model

We describe the procedure of building the matrix representing the vector model index file (section 2.2). In a classical vector space model the document is represented by the terms respectively by the weights of the terms. **In our model the document is represented not by the terms but it is represented by the common phrases (maximal phrase clusters)!**

In the previous stage of the algorithm we have identified all maximal phrase clusters - all the longest common substructures. From the definition of the phrase cluster we know that the phrase cluster is the group of documents sharing the same phrase (group of proteins sharing the same substructure). Now we can obtain the matrix representing the vector model index file directly from the generalized suffix tree. Each document (protein) is represented by the maximal phrase clusters in which it is contained. For computing the weights of the phrase clusters we used a $tf - idf$ weighting schema:

$$w_{ij} = tf_{ij} \times idf_j = tf_{ij} \times \log \frac{n}{df_j} \quad (1)$$

or

$$w_{ij} = tf_{ij} \times idf_j = (1 + \log tf(w_{ij})) \times \log(1 + \frac{n}{df_j}) \quad (2)$$

where tf_{ij} is the frequency of term t_j in document d_i and df_j is count of documents where term t_j appears in, and n is the total count of documents in collection. In some cases (see [18]) when measuring the relevance of documents returned by the system, it is suitable to use normalized form of term frequency tf_{ij}:

$$tf_{ij} = 0.5 + 0.5 \times \frac{tf_{ij}}{maxtf} \quad (3)$$

where $maxtf$ is the greatest of all tf appearing in the query.

Simple example: let's say we have a phrase cluster containing documents d_i. These documents share the same phrase t_j. We compute w_ij values for all documents appearing in a phrase cluster sharing the phrase t_j. This task is done for all phrase clusters identified by the previous stage of the algorithm.

Now we have a complete matrix representing the index file in a vector space model (section 2.2).

4.4 Building a Similarity Matrix

In the previous stage of the algorithm we have constructed a vector model index file. To build a protein similarity matrix we use standard information retrieval techniques for measuring the similarity in a vector space model. As was mentioned in section 2.2 we have used cosine similarity which looks quite suitable for our purpose. The similarity matrix will be:
Documents (proteins) similarity matrix:

$$S = \begin{pmatrix} 0 & sim(d_1, d_2) & \dots & sim(d_1, d_n) \\ sim(d_2, d_1) & 0 & \dots & sim(d_2, d_n) \\ \vdots & \vdots & \ddots & \vdots \\ sim(d_n, d_1) & sim(d_n, d_2) & \dots & 0 \end{pmatrix},$$

where the i-th row matches the i-th document (protein respectively), and the j-th column matches the j-th document (protein). The similarity matrix is diagonally symmetrical. Note that on the diagonal we have put zeros to eliminate $sim(d_i, d_i)$ which is always equal to 1 and for the simplification of the last step of the algorithm.

As this task is the most time consuming, we have developed a multi-threaded variant of computing this similarity matrix. We have simply divided the similarity matrix into n equal parts and for each n_i thread computed its own part of the similarity matrix. By this little enhancement we have achieved a very good reduction of the time needed to compute the similarity matrix - multiprocessors or multi-core processors computers required.

4.5 Finding Similar Proteins

This step is quite simple. When we have computed the similarity matrix S, we simply sort the documents (proteins) on each row according to its scores. The higher score the more similar protein is. This is done for each protein in our protein collection.

5 Evaluation and Testing

5.1 Structural Classification of Proteins

To evaluate the accuracy and effectiveness of the proposed algorithm, we compare with the SCOP database [19]. It is maintained primarily by humans in contrast with for example CATH, which uses some automated methods. In the current version of the SCOP database there are about 33000 of proteins classified. We used the SCOP because to evaluate the proposed algorithm to manually classified proteins rather than to automated methods.

There is also another structural classification system called CATH. CATH is a hierarchical classification of protein domain structures, which clusters proteins at four major levels: Class (C), Architecture (A), Topology (T) and Homologous super-family (H). The boundaries and assignments for each protein domain are determined using a combination of automated and manual procedures which include computational techniques, empirical and statistical evidence, literature review and expert analysis [2]. The CATH uses the DALI algorithm to find similarities between proteins.

5.2 Evaluation

For each protein P in our collection C, we did the following:

1. For protein P determine the class, folding pattern group, super-family, family and domain.
2. Based on the similarity matrix, find N most similar proteins P_S according to their score of similarity to protein P.
3. For each protein P_S determine the class, folding pattern group, super-family, family and domain.
4. For all proteins in our collection compute the percentage of correctly classified proteins P_S to protein P.

We did this for each protein in our collection and computed the overall percentage accuracy over our filtered collection. There are approximately 10000 unclassified proteins because they do not appear in SCOP database.

In more precise terms: let us say we have protein P. Based on the calculated similarity matrix we sort all other proteins P_S in our protein collection in descending order according to their scores. The greater the score the more similar the protein is to protein P. We take only the top N highest scoring proteins (the top N most similar proteins to the given protein). We set N to the value of 20 (in this article there are shown just first ten). After that we obtain a list such that the similar proteins for every protein in our collection we have determined the SCOP classification of those proteins. The table 1 depicts the algorithm result for the protein marked with the label 101m (SPERM WHALE MYOGLOBIN F46V N-BUTYL ISOCYANIDE AT PH 9.0). The Class with id=46456 means *All alpha proteins*, the Fold with id=46457 means *Globin-like*, the Superfamily with id=46458 means *Globin-like*, the Family with id=46463 means *Globins*, the Domain with

No.	PDB	Score	Class	Fold	SuperF	Family	Domain
	101m	**1.0**	**46456**	**46457**	**46458**	**46463**	**46469**
1	2oha	0.2573	0	0	0	0	0
2	1j52	0.2476	46456	46457	46458	46463	46469
3	1myz	0.2456	46456	46457	46458	46463	46469
4	1mz0	0.2456	46456	46457	46458	46463	46469
5	1ofk	0.2271	46456	46457	46458	46463	46469
6	106m	0.2270	46456	46457	46458	46463	46469
7	103m	0.2260	46456	46457	46458	46463	46469
8	1ofj	0.2244	46456	46457	46458	46463	46469
9	1mym	0.2001	46456	46457	46458	46463	46469
10	2ohb	0.1994	0	0	0	0	0

Table 1. Ten the most similar proteins to a protein labeled 101m and the SCOP classifications.

No.	sim 0.0	sim 0.10	sim 0.15	sim 0.20	sim 0.25
1	89.36	89.62	96.24	99.18	99.39
2	84.42	84.65	91.18	94.52	95.18
3	81.84	82.05	88.09	91.80	93.02
4	79.86	80.04	85.68	89.28	90.39
5	78.05	78.27	83.74	87.22	88.38
6	76.92	77.11	82.13	85.98	87.01
7	75.73	75.92	80.94	84.38	85.45
8	74.73	74.89	79.70	82.91	84.06
9	74.02	74.16	78.70	81.94	83.15
10	73.37	73.54	77.72	80.99	82.14
Count	32509	32297	23780	16481	11630

Table 2. Class classification percentage accuracy.

id=46469 means *Myoglobin* and finally the Species with id=46470 means *Sperm whale (Physeter catodon).*

Note that e.g. class id=0 means that the protein is not classified by the SCOP database.

The table 1 is generated for every protein in our filtered protein collection.

5.3 Experiments

We present our first results with this new method of measuring protein similarity based on their tertiary structure and the comparison with the SCOP database. All experiments were run on computer with 32 GBytes of RAM and 4 AMD 64 bit Opteron dual core CPUs. The whole PDB database indexed by our version of the suffix tree construction algorithm takes about 2.5 GBytes of RAM and about 40 minutes of time (section 4.1). The calculation of the similarity matrix 4.4 takes about 45 hours of time and 10 Gbytes of RAM since the similarity matrix is computed in memory.

First we have computed a percentage accuracy of all proteins in the entire SCOP database (32509 proteins classified), next we have computed the accuracy only for proteins for which our algorithm found proteins with at least some given score of similarity (e.g. we have protein A and for this protein exists at least one protein which has a score of similarity with protein A of at least 0.2 - we cut off all proteins which do not satisfy this assumption) - this is some kind of threshold or cutoff.

The description of the following table 2 is as follows (Figures 3, 4, 5, 6, 7 show these results in a graph representation). Column **No.** means the ordering of similar proteins (e.g. No. 1 means the most similar protein to a given protein, No. 10 means the 10th most similar protein to a given protein). Column **sim** was mentioned above. Line **Count** means for how many proteins with this cutoff were found in our collection.

In more precise terms: e.g. line 1 of the table 2 (not considering the header of the table) means that all the proteins placed in the 1st place (i.e. the most similar protein to given a protein, see the table 1) has a 89.36909% accuracy in the classification of class with no cutoff, a 89.62752% accuracy with the cutoff of proteins scoring less than 0.1, etc.

We have also identified class, fold, super-family, family and domain of proteins which are not classified by the SCOP with almost 100% membership accuracy. Table 3 shows these results. Let us examine line 4 of this table. Column $sim = 0.2$ means that we have chosen only proteins which have at least one structurally similar protein with a score of similarity of at least 0.2. Column mpa_{Class} means *minimal membership percentage accuracy to the scop protein class* (same for Fold, Superfamily, Family and Domain). Column *UPC - Unclassified proteins count* is the count of proteins which are not classified by SCOP and which appear in the first place in the list of similar proteins to a given protein. Column *TPC - Total proteins count* is the total count of proteins which have at least one structurally similar protein with a score of similarity of 0.2. In summary this means that we have found 636 unclassified proteins by using SCOP out of 16481, such that proteins have a 99.18694% class membership accuracy, a 98.87143% fold membership accuracy, etc.

6 Conclusion

In this article, we presented a novel method for measuring protein similarities using suffix tree data structure and information retrieval techniques. The method is fully automated and in comparison with the human

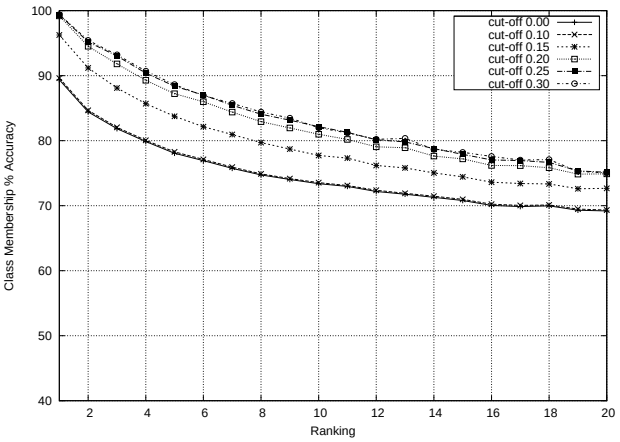

Figure 3. Protein Class Membership Percentage Accuracy.

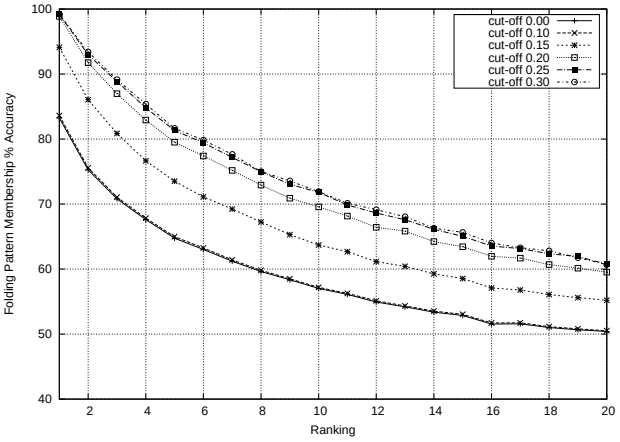

Figure 4. Protein Folding Pattern Membership Percentage Accuracy.

sim	mpa_C	mpa_F	mpa_{SF}	mpa_F	mpa_D	**UPC**	**TPC**
0.00	89.36	83.24	82.98	82.51	80.39	3352	32509
0.10	89.62	83.60	83.34	82.87	80.75	3303	32297
0.15	96.24	94.11	94.01	93.88	92.84	1395	23780
0.20	99.18	98.87	98.84	98.79	98.33	636	16481
0.25	99.39	99.19	99.19	99.15	98.85	384	11630
0.30	99.38	99.19	99.19	99.14	98.88	247	8083

Table 3. Proteins unclassified by using SCOP found by our algorithm and their membership percentage accuracy (mpa) to a given Class, Fold, Super-family, Family and Domain.

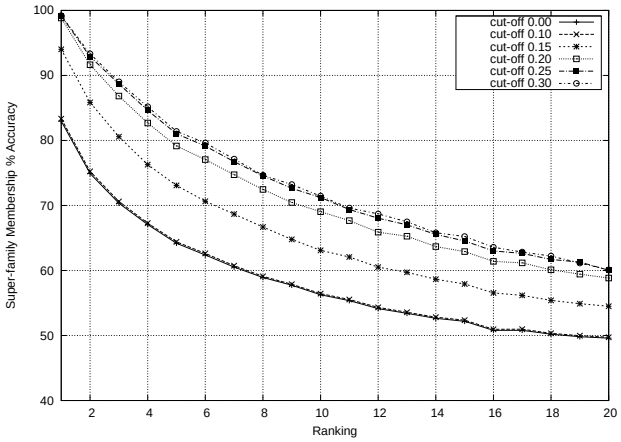

Figure 5. Protein Super-Family Membership Percentage Accuracy.

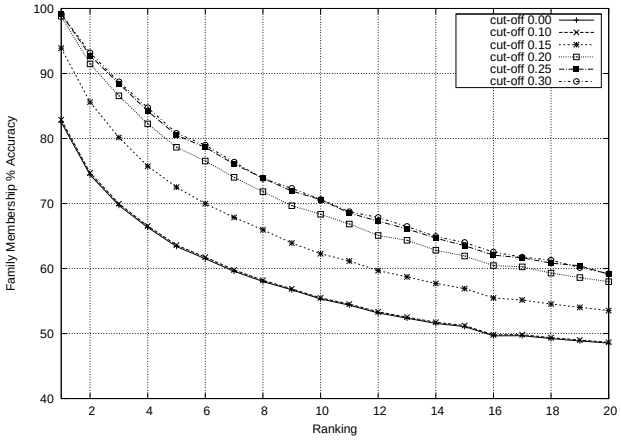

Figure 6. Protein Family Membership Percentage Accuracy.

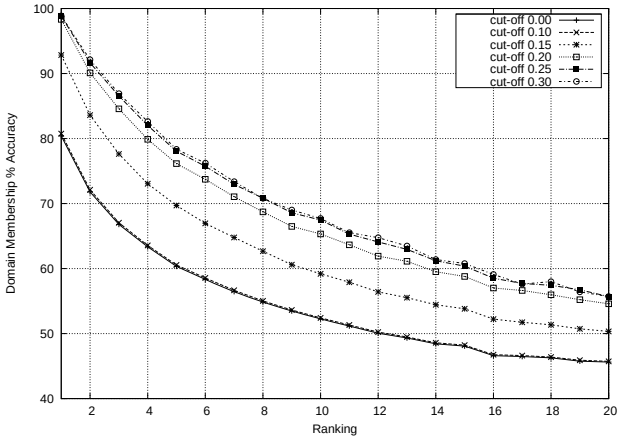

Figure 7. Protein Domain Membership Percentage Accuracy.

maintained database SCOP has achieved very good results. We have also illustrated that we can use common information retrieval models and methods for measuring similarity of proteins. With these methods we have achieved very good results.

We can also identify classes, folds, super-families, families and domains of many unclassified proteins contained in the current SCOP database with almost 100% membership accuracy. By the simple observation that when the unclassified protein is most similar to the protein which is classified and have at least some given score, than in 99% cases the unclassified protein has a similar SCOP categories as known proteins.

We have now a similarity matrix computed for all proteins included in our PDB Database. In future work we wish to use the similarity matrix for other information retrieval tasks such as clustering or application of statistical methods. The clustering of proteins is one of the first steps in the homology modeling of proteins, which we want to develop in the future. We also wish to try other methods for encoding of dihedral angles such as the clustering of these angles, which should, we believe, give better results.

References

[1] Baeza-Yates R., Ribeiro-Neto B.: Modern Information Retrieval. Adison Wesley, 1999.

[2] CATH: Protein Structure Classification http://www.cathdb.info/ (last access July-10 2009)

[3] Chim H. and Deng X.: A new suffix tree similarity measure for document clustering. In Proceedings of the 16th international Conference on World Wide Web. WWW 2007. ACM, New York, NY, pages 121-130.

[4] Ehrenfeucht A., Haussler D.: A new distance metric on strings computable in linear time. Discrete Applied Math, 20(3): 191-203 (1988).

[5] Gao F., Zaki M.J.: PSIST: Indexing Protein Structures using Suffix Trees. Proc. IEEE Computational Systems Bioinformatics Conference (CSB), pages 212-222, 2005.

[6] Google Search Engine: http://www.google.com.

[7] Gusfield, D. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.

[8] Habib T., Zhang C., Yang J.Y., Yang M.Q., Deng Y.: Supervised learning method for the prediction of subcellular localization of proteins using amino acid and amino acid pair composition. BMC Genomics 2008, 9(Suppl 1):S16.

[9] Hammouda K.M. and Kamel M.S.: Efficient phrase-based document indexing for web document clustering. IEEE Transactions on Knowledge and Data Engineering, 16(10):1279-1296, 2004.

[10] Hecker J., Yang J.Y., Cheng J.: Protein disorder prediction at multiple levels of sensitivity and specificity. BMC Genomics 2008, 9(Suppl 1):S9.

[11] Lesk A.M.: Introduction to Bioinformatics, Oxford University Press, USA, 2008.

[12] Manning, C. D.; Raghavan, P.; Schütze, H. Introduction to Information Retrieval. Cambridge University Press; 1 2008.

[13] Martinovič J., Novosád T., Snášel V.: Vector Model Improvement Using Suffix Trees. IEEE ICDIM 2007: pages 180-187

[14] McCreight E.: A space-economical suffix tree construction algorithm. In Journal of the ACM, pages 23:262–272, 1976.

[15] C.J. van Rijsbergen: Information Retrieval (second ed.). London, Butterworths, 1979.

[16] RCSB Protein Databank - PDB. http://www.rcsb.org (last access July-10 2009)

[17] Rodeh M., Pratt V.R., and Even S.: Linear algorithm for data compression via string matching. In Journal of the ACM, pages 28(1):16–24, 1981.

[18] Salton, G., and Buckley, C. Term-weighting approaches in automatic text retrieval. Information Processing and Management, 24(5):513-523, 1988.

[19] SCOP: a structural classification of proteins database for the investigation of sequences and structure. http://scop.mrc-lmb.cam.ac.uk/scop/ (last access July-10 2009)

[20] Shibuya T.: Geometric Suffix Tree: A new index structure for protein 3D structures. In Combinatorial Pattern Matching, LNCS 4009, pages 84-93, 2006.

[21] Ukkonen E.: On-line construction of suffix trees. In Algorithmica, pages 14:249–260, 1995.

[22] Wang X., Wang G., Shen C., Li L., Wang X., Mooney S.D., Edenberg H.J., Sanford J.R., Liu Y.: Using RNase sequence specificity to refine the identification of RNA-protein binding regions. BMC Genomics 2008, 9(Suppl 1):S17.

[23] Weiner P.: Linear pattern matching algorithms. In The 14th Annual Symposium on Foundations of Computer Science, pages 1–11, 1973.

[24] Yang J.Y., Yang M.Q., Dunker A.K., Deng Y., Huang X: Investigation of transmembrane proteins using a computational approach. BMC Genomics 2008, 9(Suppl 1):S7.

[25] Zamir O.: Clustering web documents: A phrase-based method for grouping search engine results. In Doctoral dissertation. University of Washington, 1999.

[26] Zamir O., Etzioni O.: Web document clustering: A feasibility demonstration. In SIGIR'98, pages 46–54, 1998.