CHAPTER 1

DISCOVERING 3-D PROTEIN STRUCTURES FOR OPTIMAL STRUCTURE ALIGNMENT

TOMÁŠ NOVOSÁD[1], VÁCLAV SNÁŠEL,[1], AJITH ABRAHAM[2] AND JACK Y YANG [3]

[1]Department of Computer Science, VŠB - Technical University of Ostrava,
17. Listopadu 15/2172, Ostrava, 70833, Czech Republic
[2]Machine Intelligence Research Labs, MIR Labs
[3]Harvard University, PO Box 400888, Cambridge, Massachusetts 02140-0888, USA

1.1 INTRODUCTION

Analyzing three dimensional protein structures is a very important task in molecular biology. Nowadays, the solution for protein structures often stems from the use of the state-of-the-art technologies such as nuclear magnetic resonance (NMR) spectroscopy techniques or X-Ray crystallography etc. as seen in the increasing number of PDB [34] entries. Protein Data Bank is a database of 3D structural data of large biological molecules, such as proteins and nucleic acids. It was proved that structurally similar proteins tend to have similar functions even if their amino acid sequences are not similar to one another. 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. [47] exploited machine learning techniques including variants of Self-Organizing Global Ranking, a decision tree, and support vector machine (SVM) algorithms to predict the tertiary structure of transmembrane proteins. Hecker et al. [14] developed a state of the art protein disorder predictor and tested it on a large protein disorder dataset

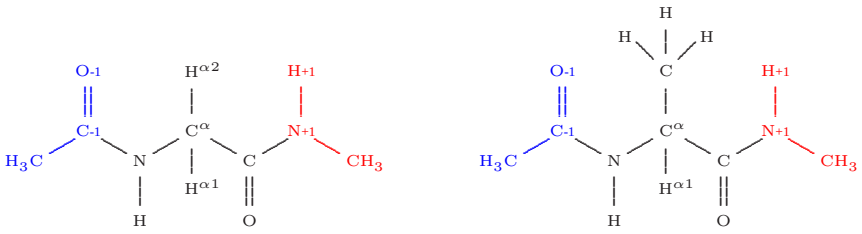
created from the Protein Data Bank. The relationship of sensitivity and specificity is also evaluated. Habib et al. [11] presented a new SVM based approach to predict the subcellular locations based on amino acid and amino acid pair composition. More protein features can be taken into consideration to improve the accuracy significantly. Wang et al. [45] 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. Another important aproach of protein structural similarity is based on database indexing methods. Gao and Zaki [9] has proposed a method for indexing protein tertiary structure by extracting a protein local feature vectors and suffix trees. Shibuya [43] developed a structure called geometric suffix tree which indexes protein 3-D structures based on their C_α atoms 3-D coordinates.

These studies are often targeted mainly at some kind of selection of the PDB database. In our past work [28, 29] we have focused on task to compute all to all protein similarities which appears in current PDB database based on their 3-D structural features. The structural similarity defined between any two proteins in PDB can be calculated using information retrieval methods and schemes and suffix trees. These methods were previously widely studied and are commonly used in these days [49, 13, 5, 23, 21]. To be able to evaluate the precision of the methods used to determine the protein structural similarity it is important to compare the results toward the existing state-of-the-art techniques or databases. The existing state-of-the-art databases of protein structural similarities are e.g. DALI [15], SCOP [42] or CATH [3].

1.2 PROTEIN STRUCTURE

Proteins are large molecules that provide structure and control reactions in all cells. In many cases only a small part of the structure - ***an active site*** - is directly functional, the rest exists only to create and fix the spatial relationship among the active site residues [19]. Chemically, protein molecules are long polymers typically containing several thousand atoms, composed of a uniform repetitive backbone (or main chain) with a particular side chain attached to each residue. The amino acid sequence of a protein records the succession of side chains. There are twenty different amino acids that make up essentially all protein molecules on earth. Every amino acid has its own original design composed of a central carbon (also called the alpha carbon - C_α) which is bonded to hydrogen, carboxylic acid group, amino group and unique side chain or R-group. The chemical properties of the R group are what give an amino acid its character.

The Danish protein chemist K.U. Linderstrøm-Lang described the protein structure in three different levels: primary structure, secondary structure and tertiary structure. For proteins composed of more than one subunit, J.D. Bernall called the assembly of the monomers the quaternary structure.



(a) Glycine (b) Alanine

Figure 1.1 Chemical structure of two amino acids.

1.2.1 Primary Structure

The unique sequence of amino acids in a protein is termed the **primary structure**. When amino acids form a protein chain, a unique bond, termed the peptide bond, exists between two amino acids. The sequence of a protein begins with the amino of the first amino acid and continues to the carboxyl end of the last amino acid. Each of the amino acid has its own unique one letter abbreviation (e.g. Alanine - A, Methionine - M, Arginine - R, ...). Thus the primary structure of the can be expressed like string of these letters. The examples of protein primary structure encoding follows:

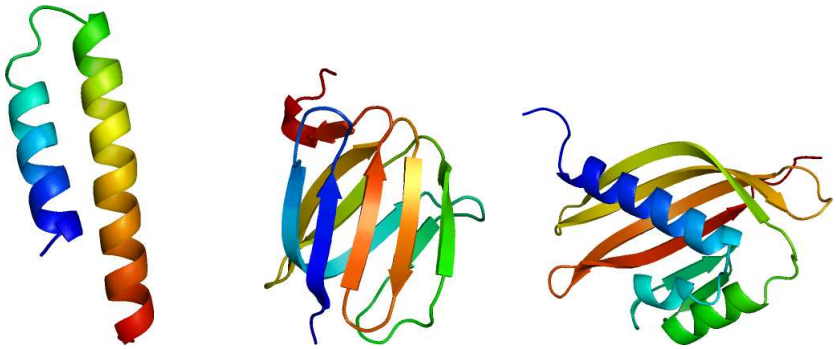
```
MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRVKHL...
MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAKSELDKAI...
AYIAKQRQISFVKSHFSRQLEERLGLIEVQAPILSRVGDGTQDNLSGAEK...
```

1.2.2 Secondary Structure

The second level in the hierarchy of protein structure consists of the various spatial arrangements resulting from the folding of localized parts of a polypeptide chain; these arrangements are referred to as **secondary structures** [20]. These foldings are either in a helical shape, called the **alpha-helix (α-helix)** (which was first proposed by Linus Pauling et. al in 1951 [32]), or a **beta-pleated sheet (β-sheet)** shaped similar to the zig-zag foldings of an accordion. The turns of the alpha-helix are stabilized by hydrogen bonding between every fourth amino acid in the chain. The beta-pleated sheet is formed by folding successive planes [35]. Each plane is five to eight amino acids long. Alpha helices and beta sheets are linked by less structured loop regions to form domains (Figure 1.2.2). The domains can potentionaly form a fully functional proteins.

1.2.3 Tertiary Structure

Tertiary structure refers to the overall conformation of a polypeptide chain that is, the three-dimensional arrangement of all its amino acid residues. Each of the atoms of amino acid residue has its own 3-D x, y, z coordinates. In contrast with secondary structures, which are stabilized by hydrogen bonds, tertiary structure is primarily stabilized by hydrophobic interactions between the non-polar side chains,



(a) α-helix (b) β-sheet (c) Domain

Figure 1.2 Secondary structure elements and domain example.

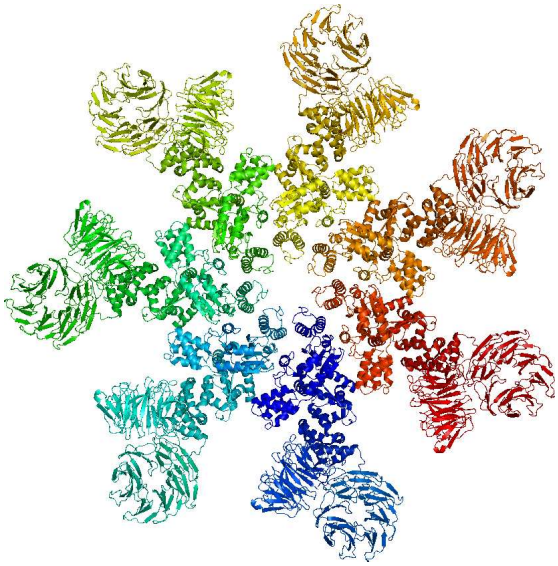


Figure 1.3 Tertiary structure of an Apoptosome-Procaspase-9 CARD complex.

hydrogen bonds between polar side chains, and peptide bonds. These stabilizing forces hold elements of secondary structure α-helices, β-strands, turns, and random coils compactly together. The most the protein structures (about 90%) available in the Protein Data Bank have been resolved by X-ray crystallography. This method allows one to measure the 3-D density distribution of electrons in the protein (in the crystallized state) and thereby infer the 3-D coordinates of all the atoms to be determined to a certain resolution. Just only about 9% of the known protein structures have been obtained by Nuclear Magnetic Resonance techniques (NMR spectroscopy) [2].

1.2.4 Quaternary Structure

Some proteins need to functionally associate with others as subunits in a multimeric structure. This is called the quaternary structure of the protein. This can also be stabilized by disulfide bonds and by non-covalent interactions with reacting substrates or cofactors. Excellent example of quaternary structure is that of hemoglobin. Adult hemoglobin consists of two alpha subunits and two beta subunits, held together by non-covalent interactions [35].

1.3 PROTEIN DATABASES

In these days there exist several protein databases publicly available on-line. These databases assembles various information about proteins, protein structures, protein functions, protein relationships, etc. Probably the main and most valuable database is the Protein Databank which consists of protein three dimensional structures resolved by state-of-the-art techniques such as X-Ray crystallography or NMR spectroscopy. Other on-line databases are generated by automated computer methods or by biologists it selfs.

1.3.1 Protein Databank - PDB

The PDB was established in 1971 at Brookhaven National Laboratory and originally contained 7 structures. Nowadays the PDB archive contains almost 80000 resolved structures and is still growing practically every day. The PDB archive is the single worldwide repository which contains information about experimentally-determined structures of proteins, nucleic acids, and complex assemblies. The structures in the archive range from tiny proteins and bits of DNA to complex molecular machines like the ribosome. The structures in this archive is resolved by the state-of-the-art methods X-Ray crystallography and NMR spectroscopy. As a member of the wwwPDB, the RCSB PDB curates and annotates PDB data according to agreed upon standards [34]. The PDB archive freely available to everyone and is updated each week at target time of Wednesday 00:00 UTC (Coordinated Universal Time). This database can be accessed online at `http://www.pdb.org`. The structures can be also downloaded from their FTP service at `ftp://ftp.wwpdb.org/pub/pdb/`.

1.3.2 SCOP: Structural Classification of Proteins

This database provides a detailed and comprehensive description of the structural and evolutionary relationships of proteins whose three-dimensional structures have been determined by X-Ray crystallography or NMR spectroscopy (PDB Databank entries). The recent version 1.75 (June 2009) of this database includes 38221 PDB entries. The classification of protein structures in the database is based on evolutionary relationships and on the principles that govern their three-dimensional structure. The method used to construct the protein classification in SCOP is essentially the visual inspection and comparison of structures though various automatic tools are used to

make the task manageable and help provide generality [42, 24, 25, 26, 27]. Each of the protein entry in SCOP database (each chain of protein respectively) is classified into the Class, Folding Pattern, Super-Family, Family, Domain and Species categories. These categories are hierarchically arranged from Class to Species. SCOP database is available with no cost for the user at `http://scop.mrc-lmb.cam.ac.uk/scop/`.

1.3.3 CATH Protein Structure Classification

The CATH is a database constructed using a semi-automatic method for hierarchical classification of protein domains [31]. The CATH stands for - **C**lass, **A**rchitecture, **T**opology and **H**omologous super-family. CATH shares many broad features with its main rival, SCOP, however there are also many areas in which the detailed classification differs greatly. CATH defines four classes: mostly-alpha, mostly-beta, alpha and beta, few secondary structures. Much of the work in CATH database is is done by automatic methods toward the SCOP, though there are important manual tasks to the classification. The Most important step in CATH classification is to separate the proteins into domains. The domains are next automatically sorted into classes and clustered on the basis of sequence similarities. These clusters (groups) form the **H levels** of the classification (homologous super-family groups). The topology level is formed by structural comparisons of the homologous groups. Finally, the Architecture level is assigned manually [31]. For more detailed descriptions of CATH database building process and comparison with SCOP and other databases please see [12, 6]. CATH database can be accessed and searched at `http://www.cathdb.info/`.

1.3.4 DALI - Distance matrix ALIgnment

The DALI database is based on exhaustive all-against-all 3D structure comparison of protein structures currently in the PDB. The structural neighborhoods and alignments are automatically maintained and regularly updated using the DALI search engine. The DALI algorithm works with 3-D coordinates of each protein that are used to calculate residue-to-residue (C_α-to-C_α) distance matrices. The distance matrices are first decomposed into elementary contact patterns, e.g. hexapeptide-hexapeptide submatrices. Then, similar contact patterns in the two matrices are paired and combined into larger consistent set of pairs. This method is fully automatic and identifies structural resemblances and common structural cores accurately and sensitively, even in the presence of geometrical distortions [15, 16]. The DALI database can be accessed from the DALI server at `http://ekhidna.biocenter.helsinki.fi/dali`.

1.4 VECTOR SPACE MODEL

The vector model [1] of documents was established in the 1970's [37, 38]. A document in the vector model is represented as a vector. Each dimension (element) of this vector corresponds to a separate term appearing in document collection. If a term occurs in the document, its value in the vector is non-zero. The vector model is widely used

information retrieval scheme for measuring similarity between documents it selfs or between user query and documents in the collection [29, 28, 23, 7, 17, 41, 39, 18].

In the vector model there are m different terms $t_1, \ldots, t_m$ for indexing N documents. Then each document d_i is represented by a vector:

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

where w_{ij} is the weight of the term t_j in the document d_i. These term weights are ultimately used to compute the degree of similarity between each document stored in the system and the user query. The weight of the term in the document vector can be determined in many ways. A common approach uses the so called $tf \times idf$ (Term Frequency $\times$ Inverse Document Frequency) method [40], in which the weight of the term is determined by these factors: how often the term t_j occurs in the document d_i (the term frequency tf_{ij}) and how often it occurs in the whole document collection (the document frequency df_j. Precisely, the weight of the term t_j in the document d_i is [18]:

$$w_{ij} = tf_{ij} \times idf_j = tf_{ij} \times \log \frac{n}{df_j} \tag{1.1}$$

where idf stands for the inverse document frequency. This method assigns high weights to terms that appear frequently in a small number of documents in the document set.
An index file of the vector model is represented by matrix:

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

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

The similarity of two documents in vector model is usually given by the following formula – Cosine Similarity Measure:

$$sim(d_i, d_j) = \cos\theta = \frac{\sum_{k=1}^{m} (w_{ik} w_{jk})}{\sqrt{\sum_{k=1}^{m} (w_{ik})^2 \sum_{k=1}^{m} (w_{jk})^2}} \tag{1.2}$$

Suppose we have two documents $d_1 = (w_{11}, w_{12})$ and $d_2 = (w_{21}, w_{22})$, where w_{11}, w_{12} represent the weights of terms t_1, t_2 in document d_1 and w_{21}, w_{22} represent the weights of terms t_1, t_2 in document d_2. Then the geometrical representation of cosine similarity is shown in Figure 1.4.

For more information about vector space model, please consult [22, 33, 1, 37, 38, 39].

1.5 SUFFIX TREES

A suffix tree is a data structure that allows efficient string matching and querying. Suffix trees have been studied and used extensively, and have been applied to fun-

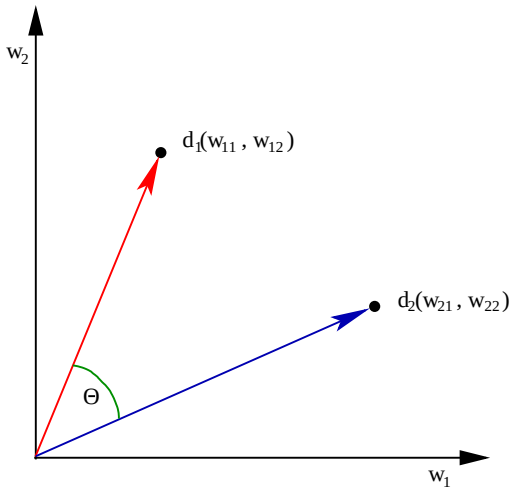


Figure 1.4 Geometrical representation of cosine similarity.

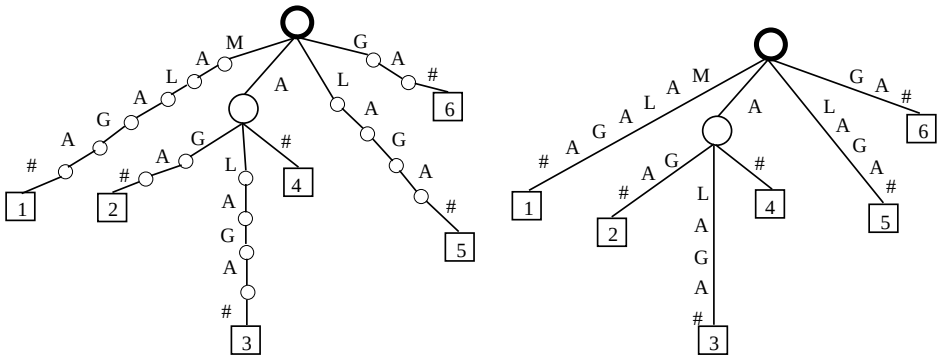
damental string problems such as finding the longest repeated substring [46], strings comparisons [8], and text compression [36]. Following this, we describe the suffix tree data structure - its definition, construction algorithms and main characteristics.

1.5.1 Definitions

The following description of the suffix tree was taken from Gusfield's book *Algorithms on Strings, Trees and Sequences* [10]. Suffix trees commonly dealing with strings as sequence of characters. 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. Citation [48]:

Definition 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 $S[i \ldots m]$.

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



(a) Suffix Trie (b) Suffix Tree

Figure 1.5 Simple example of suffix trie and suffix tree of string `MALAGA#`.

Suppose we have a short protein sequence `MALAGA` which is combination of four amino acids - Methionine, Alanine, Leucine and Glycine. Example of suffix trie of the string `MALAGA#` is shown in Figure 1.5.1 (a). Corresponding suffix tree of the string `MALAGA#` is presented in Figure 1.5.1 (b). There are six leaves in this example, marked as rectangles and numbered from 1 to 6. The terminating characters are also shown in this Figure.

In a similar manner, a suffix tree of a set of strings, called a generalized suffix tree [10], is a compact trie of all the suffixes of all the strings in the set [48]:

Definition 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 $S_i[j \ldots m_i]$.

Figure 1.6 is an example of a generalized suffix tree of the set of two strings - RNADNA# and DNARNA#. The internal nodes of the suffix tree are drawn as circles, and are labeled from *a* to *d*. Leaves are drawn as rectangles and numbers $d_i = (d_1, \ldots, d_n)$ in each rectangle indicates the string from which that suffix originates - a unique number that identifies the string. Each string is considered to have a unique terminating symbol.

1.5.2 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 $S[1 \ldots L]$ into the tree. Then it successively enters the suffix $S[i \ldots L]$ into the growing tree

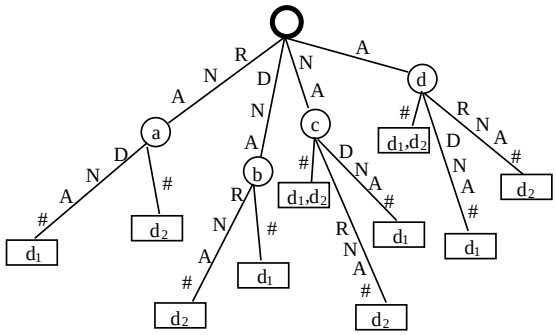


Figure 1.6 Example of the generalized suffix tree.

for i increasing from 2 to L. The details of this construction method are not within the bounds of this article. Various suffix tree construction algorithms can be found in [10] (a good book on suffix tree construction algorithms in general).

Several linear time algorithms for constructing suffix trees exist [30, 44, 46]. 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 naive 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 for finding an appropriate child of the root (this is important because the root node has the same count of child nodes as it is the size of the alphabet - count of terms in document collection) and logarithmic time to find an existing child or to insert a new child node to any other internal nodes of the tree [23]. Next we have also improved the generalized suffix tree data structure to be suitable for large document collections [23].

1.6 INDEXING 3-D PROTEIN STRUCTURES

As was mentioned above the data for protein 3-D structures indexing is retrieved from PDB database, which consists of proteins, nucleic acids and complex assemblies. Before indexing protein structures we consider only complete protein structures. We filter out all nucleic acids and complex assemblies from the entire PDB database. Next we filter 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 a collection of files containing a description of a specific protein and its three dimensional structure and containing only amino acid residues with complete

a N-Cα-C-O atom sequence. Each retrieved file has at least one main chain (some proteins have more than one main chain) of at least one model (some PDB files contained more models of three dimensional protein structure). In cases when the PDB file contained multiple chains or models, we took into account all of those (all main-chains of all models).

To be able to measure 3-D protein structure similarity using suffix trees and vector space model, we need to encode protein 3-D structure into a vector. representation.

1.6.1 Torsion 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 torsion angle can be defined by four, pairwise non-collinear vectors.

The backbone torsion 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 torsion angles tend to be from -180° to 180°.

1.6.2 Encoding the 3-D Protein Main Chain Structure for Indexing

To be able to index proteins by information retrieval (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). The area of protein 3D structure encoding has been widely studied by authors in previous works e.g. [50, 9, 4]. 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 example let us say the protein backbone consists of six amino acid residues RNADNA (abbreviations for Arginine, Asparagine, Alanine and Aspartic acid). The relationship between the two following residues can be described by its torsion angles ϕ, ψ and ω. Since ϕ and ψ are taking values from the interval $\langle -180°, 180° \rangle$ it must be done some normalization. From this interval can be obtained discreet values by dividing the interval into equal sized subintervals (for example into 36 subintervals), e.g. $-180°$, $-170°$,...,$0°$,$10°$,...,$180°$. Each of these values was labeled with non-negative integers as follows: 00, 01,..., 36 where 00 stands for $-180°$. Now, let's say that ϕ is $-21°$, the closest discrete value is $-20°$ which has the label 16, so we have encoded this torsion with the string '16'. The same holds for ψ. Torsion angle 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 $\omega \approx 180°$, $\phi \approx -170°$, $\psi \approx -160°$. Concatenation was done in the following manner: $\omega\phi\psi$.

1.6.3 Indexing

The objective of this stage is to prepare the data for indexing by suffix trees. The suffix tree can index sequences. The resulting sequence in this case is a sequence of nonnegative integers. For example, let's say we have a protein with a backbone consisting of 6 residues e.g. RNADNA with its three dimensional properties. The resulting encoded sequence can be for example:
{A3202, A2401, A2603, A2401, A2422, A2422, A2220}
After obtaining this sequence of 6 words, we create a dictionary of these words (each unique word receives its own unique non negative integer identifier). The translated sequence appears as follows:
{0, 1, 2, 1, 3, 3, 4}
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.

1.7 PROTEIN SIMILARITY ALGORITHM

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

1. Encode 3-D protein structure into vectors 1.6.
2. Insert all encoded main chains of all proteins in the collection into the generalized suffix tree data structure.
3. Find all maximal substructure 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.

1.7.1 Inserting All Main Chains into the Suffix Tree

At this stage of the algorithm, we construct a generalized suffix tree of all encoded main chains. As mentioned in Section 1.6, we obtain 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 1.5).

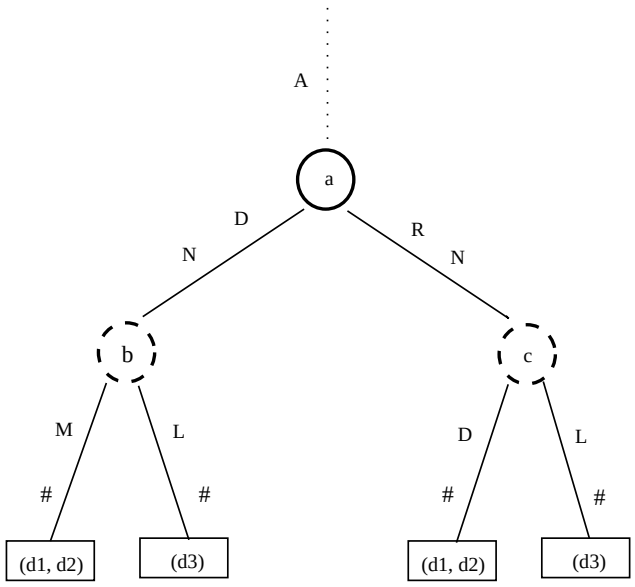


Figure 1.7 Example of maximal phrase cluster - node *b* and *c*.

1.7.2 Finding All Maximal Substructure Clusters

To be able to build a vector model of proteins, we have to find all maximal phrase clusters. Recall the example given in Section 1.6: the phrases can be e.g. RNADNA#, RNA#, DNA#, etc. (just imagine that phrase RNA# is equal to *"A3202 A2401 A2603 #"*). 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) [49]:

Definition 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). Maximal phrase cluster can be seen as a kind of 3-D structural alignment - common parts of 3-D protein structure shared between two or more proteins. Figure 1.7 displays the example of maximal phrase cluster.

1.7.3 Building a Vector Model

We describe the procedure for building the matrix representing the vector model index file (section 1.4). In a classical vector space model, the document is represented by the terms (which are words) respectively and by the weights of the terms. **In our model the document is represented not by the terms but by the common phrases (maximal phrase clusters)!** - the term in our context is a common phrase i.e. maximal phrase cluster.

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 the 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 are using a $tf \times idf$ weighting schema as given by Equation 1.1.

Simple example: Let us say that 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 1.4).

1.7.4 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 mentioned in Section 1.4, we have used cosine similarity, which looks quite suitable for our purpose. The similarity matrix is given by:
Documents (proteins) similarity matrix:

$$S = \begin{pmatrix} 1 & sim(d_1, d_2) & \dots & sim(d_1, d_n) \\ sim(d_2, d_1) & 1 & \dots & sim(d_2, d_n) \\ \vdots & \vdots & \ddots & \vdots \\ sim(d_n, d_1) & sim(d_n, d_2) & \dots & 1 \end{pmatrix},$$

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.

1.7.5 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 their similarity scores. The higher the score, the more similar the 2 proteins are. This is done for each protein in our protein collection.

1.8 SUMMARY

REFERENCES

1. Baeza-Yates R., Ribeiro-Neto B.: Modern Information Retrieval. Adison Wesley, 1999.
2. Bünger, A.T.: X-PLOR, Version 3.1. A System for X-ray Crystallography and NMR. Yale University Press, New Haven, USA, 1992.
3. CATH: Protein Structure Classification
   http://www.cathdb.info/ (last access July-10 2009)
4. Chew, L. P. Huttenlocher, D. Kedem, K. Kleinberg, J.: Fast Detection of Common Geometric Substructure in Proteins, Journal of Computational Biology, Vol. 6, No. 3/4, pages 313-326, 1999.
5. 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.
6. Day R., Beck D. A., Armen R. S., Daggett V.: A consensus view of fold space: Combining SCOP, CATH, and the Dali Domain Dictionary. Protein Sci. 12 (10), pp. 2150–2160, 2003.
7. Dráždilová, P., Dvorský, J., Martinovič, J., Snášel, V.: Search in Documents based on Topical Development. In AWIC 2009 Amazon.ca, 2009.
8. Ehrenfeucht A., Haussler D.: A new distance metric on strings computable in linear time. Discrete Applied Math, 20(3): 191-203 (1988).
9. Gao F., Zaki M.J.: PSIST: Indexing Protein Structures using Suffix Trees. Proc. IEEE Computational Systems Bioinformatics Conference (CSB), pages 212-222, 2005.
10. Gusfield, D. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.
11. 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.
12. Hadley C., Jones D. T.: A systematic comparison of protein structure classifications: SCOP, CATH and FSSP. Structure 7 (9): pp. 1099–1112, 1999.
13. 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.
14. Hecker J., Yang J.Y., Cheng J.: Protein disorder prediction at multiple levels of sensitivity and specificity. BMC Genomics 2008, 9(Suppl 1):S9.

15. Holm, L. Sander, C.: Dali: a network tool for protein structure comparison. In Trends in Biochemical Sciences, Vol 20; Number 11, 1995.
16. Holm, L. Sander, C.: Protein structure comparison by alignment of distance matrices. J. Mol. Biol., 233, pp. 123-138, 1993.
17. Hruška P., Martinovič J., Dvorský J., and Snášel V.: XML Compression Improvements Based on the Clustering of Elements, CISIM, Poland, 2010.
18. Lee, D. L., Chuang, H., Seamons, K. E.: Document ranking and the vector-space model. In *IEEE Software*, pages 67–75, 1997.
19. Lesk A.M.: Introduction to Bioinformatics, Oxford University Press, USA, 2008.
20. Lodish, H., Berk, A., Matsudaira, P., Kaiser, A. C., Krieger, M., Scott, P. M., Zipursky, L., Darnell, J.: Molecular Cell Biology. 6th Edition, W. H. Freeman, 2007.
21. Lexa M., Snášel V., Zelinka I.: Data-mining protein structure by clustering, segmentation and evolutionary algorithms. In Data Mining: Theoretical Foundations and Applications. Germany : Springer Verlag, 2009. Studies in Computational Intelligence, Volume 204, ISBN 978-3-642-01087-3, pp. 221-248
22. Manning, C. D.; Raghavan, P.; Schütze, H. Introduction to Information Retrieval. Cambridge University Press; 1 2008.
23. Martinovič J., Novosád T., Snášel V.: Vector Model Improvement Using Suffix Trees. IEEE ICDIM 2007: pages 180-187
24. Murzin A. G., Brenner S. E., Hubbard T., Chothia C.: SCOP: a structural classification of proteins database for the investigation of sequences and structures. J. Mol. Biol. 247, 536-540, 1995.
25. Lo Conte L., Brenner S. E., Hubbard T.J.P., Chothia C., Murzin A.: SCOP database in 2002: refinements accommodate structural genomics. Nucl. Acid Res. 30(1), 264-267, 2002.
26. Andreeva A., Howorth D., Brenner S.E., Hubbard T.J.P., Chothia C., Murzin A.G.: SCOP database in 2004: refinements integrate structure and sequence family data. Nucl. Acid Res. 32:D226-D229, 2004.
27. Andreeva A., Howorth D., Chandonia J.-M., Brenner S.E., Hubbard T.J.P., Chothia C., Murzin A.G.: Data growth and its impact on the SCOP database: new developments. Nucl. Acids Res. 36: D419-D425, 2008.
28. Novosád T., Snášel V., Abraham, A., Yang, J.Y.: Prosima - Protein similarity algorithm. World Congress on Nature and Biologically Inspired Computing, pages 84-91, 2009.
29. Novosád T., Snášel V., Abraham, A., Yang, J.Y.: Searching Protein 3-D Structures for Optimal Structure Alignment Using Intelligent Algorithms and Data Structures. In IEEE Transaction on Information Technology in Biomedicine, pages 1378-1386, 2010.
30. McCreight E.: A space-economical suffix tree construction algorithm. In Journal of the ACM, pages 23:262–272, 1976.
31. Orengo, C. A., Michie A. D., Jones S., Jones, D. T., Swindells, M. B., Thornton J. M.: CATH-a hierarchic classification of protein domain structures. Structure 5 (8), pp. 1093–1108, 1997.
32. Pauling L., Corey, R.B., Branson, H.R.: The structure of proteins; two hydrogen-bonded helical configurations of the polypeptide chain. Proc Natl Acad Sci USA 37, Vol. 4, pp. 205–211, 1951.

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

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

35. Robinson R.: Genetics. M

36. 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.

37. Salton, G.: The SMART Retrieval System - Experiments in Automatic Document Processing. Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

38. Salton, G., Lesk, M. E.: Computer evaluation of indexing and text processing. Journal of the ACM, 15(1):8-36, 1968.

39. Salton, G., Wong, A., Yang, C. S.: A vector space model for automatic indexing. Communications of the ACM, Vol. 18, Issue 11, 1975.

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

41. Sarkar, I. N.: A vector space model approach to identify genetically related diseases. J Am Med Inform Assoc, 2012.

42. 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)

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

44. Ukkonen E.: On-line construction of suffix trees. In Algorithmica, pages 14:249–60, 1995.

45. 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.

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

47. 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.

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

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

50. Ye et al.: Pairwise protein structure alignment based on an orientation-independent backbone representation, Journal of Bioinformatics and Computational Biology 2, pp. 699–718, 2004.