Clustering of Protein Sequences

Ivana Rudolfová*
rudolfa@fit.vutbr.cz

Jaroslav Zendulka

Abstract: Clustering of protein sequences is one of the techniques that can be helpful for predicting secondary structure of protein. Clustering methods are based on expressing similarity or dissimilarity of such sequences. The similarity of two protein sequences can be assessed by the score of the best alignment of the sequences. The paper deals with using formal model theory in describing this important task of bioinformatics. First, a well-known Needleman and Wunsch algorithm for finding the best alignment is introduced. Then a finite automaton that can compute the score for the best alignment is proposed.

Key Words: protein sequence, optimal alignment of two protein sequences, finite automaton, clustering, similarity function

1 Introduction

Prediction of local structure of proteins from amino acid sequence is one of the most interesting tasks in bioinformatics. The structure of a protein determines its function, so knowing the structure of proteins we can infer their function. The structure of a protein can be determined by the X-ray crystallography. However, this experiment is quite expensive and time consuming and the isolated protein is necessary for it. Due to the large number of sequenced mRNA there exist many protein sequences in specialized databases, which were translated from mRNA sequences but they have never been isolated. For such proteins, the X-ray crystallography cannot be used to determine the 3D structure of the macromolecule so the only way how to infer the function of protein is to try to predict the structure.

Christopher Bystroff and David Baker proposed a new method for prediction of local structure in proteins [1]. They clustered the set of recurrent amino acid sequence patterns and they found that many of the sequence patterns occur primarily in a single type of local structure. The similarity of the sequence patterns can be expressed in terms of the score of the best alignment of sequences being compared.

The paper deals with using formal model theory in describing the problem of computing the score of the best alignment of two sequences. A well-known Needleman and Wunsch algorithm for finding the best alignment and its adaptation to the protein sequence alignment is introduced in section 3.1. A finite automaton that can be used to find the optimal alignment of two protein sequences is proposed in section 3.2. Finally the process of clustering of amino acids sequences is described in the terms of mathematical functions in section 4.

2 Background

Proteins are complex macromolecules that consist of amino acids joined by peptide bonds. The twenty amino acids found in proteins have similar chemical structures, varying only in the chemical group attached to the α-carbon (R group). The varying R group is called the side

* Faculty of Information Technology, Bozetechova 2, CZ-612 66 Brno

chain. One-letter code is assigned to each amino acid. The order in which the various amino acids are assembled into a protein is the sequence, or primary structure, of the protein. Proteins quickly fold into a compact form in living organism. There exist several methods for predicting the 3D structure of protein, one of them uses clustering of protein sequences.

Clustering is a type of unsupervised learning approach aiming to divide a group of objects into clusters (classes) such a way that similarity of objects belonging to the same class is maximized whereas similarity of objects from different classes is minimized. By this approach new patterns and groupings can readily be identified.

In the case of amino acid sequences Ch. Bystroff and D. Baker used clustering to create the groups of similar recurrent amino acid sequence patterns [1]. They got 82 groups and studied the structural properties of the sequences. They found that the sequences in one group occur primarily in one or two types of secondary structural elements. Finally, they created a database of these groups and described the structural properties of the most favorable conformations (referred to as paradigm sequences) and computed the confidence of each conformation there. This database can be used for prediction of secondary structure of a protein. If a sequence contains a pattern from the database, the corresponding part of the sequence can fold in the described structure with some confidence (due to the similarity of the pattern and the paradigm sequence in a database)

Clustering methods use a similarity or dissimilarity function to compare two objects. The similarity of two amino acid sequences of the same length can simply be computed by comparing amino acids at each position. The similarity of any two amino acids can be determined as a bonus for match or penalty for non-match using so called PAM or BLOSUM matrices, which will be described later. If the length of the sequences is different, similarity can be expressed as the score of optimal alignment. Finding the optimal alignment is a task, which is often used in bioinformatics when comparing nucleic or amino acid sequences.

3 Optimal Alignment of Sequences

The alignment of two sequences is a pairwise match between the characters of each sequence. In the simplest case (no internal gaps) aligning two sequences is simply a matter of choosing the starting point for the shorter sequence. At any given position within a sequence three kinds of changes can occur: (1) a mutation that replaces one character with another, (2) an insertion that adds one or more positions, or (3) a deletion that deletes one or more position. To reflect the occurrence of insertions and deletions, gaps are commonly added in alignments. Consideration of the possibility of insertion and deletion events significantly complicates sequence alignments by vastly increasing the number of possible alignments between two sequences. To find the optimal alignment for two sequences, it is necessary to decide how to evaluate each alignment (score). The scoring function is determined by the amount of credit an alignment receives for each aligned pair of identical amino acids (the match score), the penalty for aligned pairs of nonidentical amino acids (the mismatch score) and the penalty for the gap (gap penalty). A simple alignment score for a gapped alignment of sequences u and v can be computed as follows:

$$s(u', v') = \sum_{i=1}^{n} \begin{cases} gap\ penalty;\ if\ u'_i = '-'\ or\ v'_i = '-' \\ match\ score;\ if\ no\ gaps\ and\ u'_i = v'_i \\ mismatch\ score;\ if\ no\ gaps\ and\ u'_i \neq v'2_i \end{cases}$$

where n is the length of the longer sequence.

When aligning two sequences of amino acids it is desirable to use different mismatch score for different pairs of aligned amino acids. This non-uniform mismatch score better reflects the fact that some substitutions of amino acids are more common in nature than others and the fact, that some substitutions affect the properties of the protein more than others. The properties of a protein are determined by the side chains of the constituent amino acids. The side chains of two amino acids can be very similar, while the side chain of other two amino acids can be quite different. To cover all these facts by the mismatch score, the scoring matrices are commonly used to determine the score of the alignment. There are two families of scoring matrices for aligning protein sequences – PAM matrices and BLOSUM matrices. They were derived by observing substitution rates among the various amino acid in nature and they contain the match and the mismatch score for each pair of amino acids (the match score can be also different for different pairs of amino acids).The example of scoring matrix is shown in Figure 1. (This matrix will be used in our similarity function).

A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	12															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	-2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6					
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V

Figure 1: Matrix PAM (250)

3.1 The Needleman and Wunsch Algorithm

When we know how to evaluate the alignment of two sequences, we can search for an algorithm that finds the best alignment of alignments between two sequences. The most obvious method, exhaustive search of all possible alignments, is generally not feasible. As the lengths of the sequences grow, the number of possible alignments to search quickly becomes intractable, or impossible to compute in a reasonable amount of time. This problem can be overcome by using dynamic programming, a method of breaking a problem apart into reasonably sized subproblems, and using these partial results to compute the final answer. S.

Needleman and C. Wunsch were the first who applied a dynamic programming approach to the problem of sequence alignment [2].

When we align two sequences, there are three possibilities for the first position in our alignment: (1) we can place a gap in the first sequence, (2) place a gap in the second sequence, or (3) place a gap in neither sequence. For the first two cases the alignment score for the first position will equal the gap penalty, while the rest of the score will depend on how we align the remaining parts of each sequence. For the last case the alignment score for the first position will equal the value found in the scoring matrix (e.g. PAM) for the first symbols of the sequences. Again, the rest of the score will depend on how we align the remaining sequences. By this way we can breakdown the problem of finding the optimal alignment. If we knew the score for the best alignment of the remaining sequences, we can determine the best alignment of the first position.

The dynamic programming algorithm computes optimal sequence alignments by filling in a table of partial sequence alignment scores until the score for the entire sequence alignment has been calculated. The algorithm utilizes a table in which the horizontal and vertical axes are labeled with the two sequences to be aligned. Figure 2 illustrates the partial alignment of two sequences *AVPT* and *AILVPT* for gap penalty = -1 and the match score for two Alanines (A) equal 2.

		A	V	P	T
	0	-1	-2	-3	-4
A	-1	2			
I	-2				
L	-3				
V	-4				
P	-5				
T	-6				

Figure 2: A partial score table for aligning sequences *AVPT* and *AILVPT*.

The alignment of the two sequences is equivalent to a path from the upper left corner of the table to the lower right. A horizontal move in the table represents inserting a gap into the sequence along the left axis. A vertical move represents inserting a gap into the sequence along the top axis, and a diagonal move represents the alignment of the amino acids of both sequences. As the outset of the algorithm, the first row and column of the table are initialized with multiples of the gap penalty, as shown in figure 2. With the algorithm we begin filling in the table with position (2,2), the second entry in the second row. This position represents the first column of our alignment. Because we have three possibilities for this first position (a gap in the first or in the second sequence, or alignment of amino acids) we can fill the first position in the table with one of three possible values:

1. We can take the value from the left (2,1) and add the gap penalty, representing a gap in the sequence along the left axis;
2. We can take the value from above (1,2) and add the gap penalty, representing a gap in the sequence along the top axis; or
3. We can take the value from the diagonal element above and to the left (1,1) and add the match bonus or mismatch penalty for the two nucleotides along the axes, representing an alignment of the two amino acids.

To fill in the table, we take the maximum value of these three choices. Once we have position (2,2) filled, we can fill in the rest of row 2 in a similar manner, followed by row 3, and

likewise for the rest of the table. The Figure 3 shows the completion of the partial scores table for sequences *AVPT* and *AILVPT* (with gap penalty -1 and the score for match and mismatch from scoring matrix PAM (250) in Figure 1).

		A	V	P	T
	0	-1	-2	-3	-4
A	-1	2	1	0	-1
I	-2	1	6	5	4
L	-3	0	5	4	3
V	-4	-1	4	4	4
P	-5	-2	3	10	9
T	-6	-3	2	9	13

Figure 3: The completion of the partial score table for sequences *AVPT* and *AILVPT*.

Once the table has been completed, the value in the lower right represents the score for the optimal gapped alignment between two sequences. For our example, the optimal alignment between the sequences *AVPT* and *AILVPT* has the score 13. The alignments are:

A - - V P T A V - - P T

A I L V P T A I L V P T

These optimal alignments could be found by reconstructing the paths from the lower rightmost cell of the the partial score table to the upper leftmost one. But to express similarity, it is enough to know the score without necessity to care, for which alignment the score was achieved.

3.2 A Finite Automaton for the Optimal Alignment

In this section the special type of finite automaton, which can compute the score of the optimal alignment between two sequences of amino acids, is described. First, the definition of a deterministic finite automaton is reminded:

The deterministic finite automaton is 5-tuple $M = (Q, \Sigma, \delta, g_0, F)$, where

Q is a finite set of the states

Σ is a finite input alphabet

δ is a transition function of the form $\delta : Q \times \Sigma \rightarrow Q \cup \{undef\}$, such that $|\delta(g,a)| = 1, \forall g \in Q$

$g_0 \in Q$ is an initial state

$F \subseteq Q$ is a set of finite states.

Originally, the finite automaton is a machine that sequentially reads symbols from an input tape and changes its state according to the input symbol (transition function). It is usually used for determining if a string belongs to a specified language. The finite automaton accepts the string, if it reaches the finite state after reading all symbols from the input tape. All strings accepted by specified finite automaton define the language, which is described by this automaton.

To use the finite automaton for computing the score of the best alignment of two sequences of amino acids we have to slightly modify its definition. Our automaton will compute the score by using the previously described algorithm for finding the best alignment of two sequences (simulation of filling the partial score table). Necessary modifications in automaton behavior are the following:

1. The automaton should work with two input tapes.

2. The automaton should make it possible to perform transitions in parallel, since we are not able to say which alignment will be the best one at the beginning of the simulation.

3. The transition function will not depend on input symbols. At each state, there are three possibilities what can be done – insertion of a gap in first or second sequence, or alignment of corresponding symbols. As a result, there are three transitions form each state, which should be done in parallel. The input symbols are only used for computing the partial score.

4. Each inner position in the partial scores table can be reached by three ways, and the one, which produce the best score, is chosen. In the finite automaton we need three states to simulate these possibilities. From these three states, the one with the best score should be determined for the next simulation. It is useless to proceed the simulation from the other two states, so the simulation from these states can be terminated.

The Figure 4 shows how the modified automaton will look like.

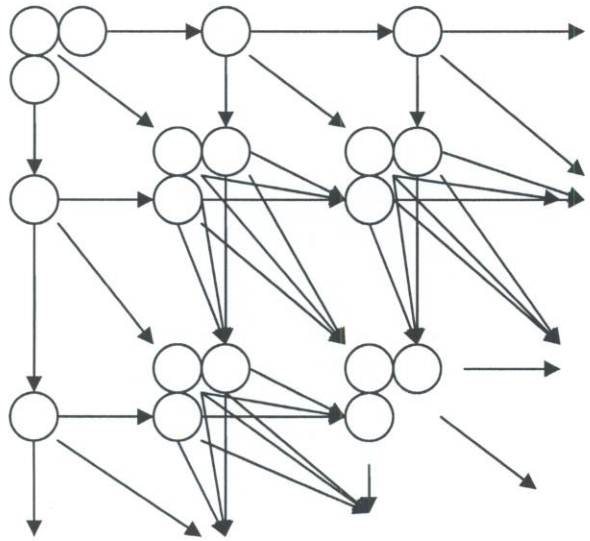

Figure 4: Finite state automaton for computing the score of the best alignment of two sequences of amino acids.

The states in the automaton can be denoted by labels, which correspond to positions in the partial scoring table and to the direction, from which the state was reached – e.g. (2,2,v) denotes the state reached from the state (2,1,h) by vertical transition, (2,2,h) denotes the state reached from the state (1,2,v) by horizontal transition, and (2,2,d) denotes the state reached from the state (1,1) by diagonal transition. The exception is the initial state, which we denote only by the position: (1,1). There are three types of transitions in the automaton: horizontal, vertical and diagonal. The horizontal transition corresponds to reading a symbol from the first sequence (inserting a gap in the second sequence), the vertical transition corresponds to reading a symbol from the second sequence (inserting the gap in the first sequence), and the diagonal transition corresponds to reading symbols from both sequences (aligning of the symbols). Each state contains the value of the partial score. The score is computed from the partial score of the previous state by adding a gap penalty (horizontal and vertical transitions), mismatch penalty (diagonal transitions with different symbols in the sequences) or match bonus (diagonal transitions with the same symbols in the sequences). According to this value, the best state at each position is determined and the simulation proceeds from this state. The transitions from the other two states at the position are not performed so the value of each

state is unambiguously defined. The finite state of the automat is one of the states *(m,n,h)*, *(m,n,v)*, or *(m,n,d)* with the highest score, where *m* and *n* are the lengths of the sequences.

In conclusion, the finite automaton for computing the score of the best alignment of two sequences of amino acids of the length *m* and *n* can be described mathematically as follows:

The score-computing automaton is 5-tuple $S=(Q,\Sigma,\delta,g_0,F)$, where

Q is a finite set of states, $|Q|=((m*n)*3)+m+n+1$,

$$Q=\begin{Bmatrix}(1,1),(2,1,h),(3,1,h),\ldots,(m+1,1,h),(1,2,v),(1,3,v),\ldots,(1,n+1,v),(2,2,h),(2,2,v),(2,2,d),\ldots,\\(m+1,n+1,h),(m+1,n+1,v),(m+1,n+1,d)\end{Bmatrix}$$

each state has a value c, which is defined recursively as follows: Let

$mV(x,y)=\max\{c(x,y,h):c(x,y,h)\in Q\}\cup\{c(x,y,v):c(x,y,v)\in Q\}\cup$
$\{c(x,y,d):c(x,y,d)\in Q\}$

for $1\le x\le m$ and $1\le y\le n$ be the maximum c-value of states from Q the name of which starts with x,y. Let $\max\{\ \}=0$. Then $c(1,1)=0$ and $c(x+1,y,h)=c(x,y+1,v)=mV(x,y)-gp$, $c(x+1,y+1,d)=mV(x,y)+M(a_x,b_y)$ for all other states from Q. Here, gp is a gap penalty, a_x is the xth character in the first sequence and b_y is the yth character from the second sequence, and $M(a_x,b_y)$ is an entry with indexes a_x and b_y in a scoring matrix (e.g. PAM 250) used for computing the partial score.

Σ is a finite input alphabet of all amino acids,
$\Sigma=\{A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V\}$

δ is a transition function of the form $\delta:Q\to Q\times Q\times Q$

$g_0=(1,1)\in Q$ is an initial state

$$F=\begin{Bmatrix}(m+1,n+1,z)\mid z\in\{h,v,d\}\wedge c(m+1,n+1,z)=\\\max\{c(m+1,n+1,h),c(m+1,n+1,v),c(m+1,n+1,d)\}\end{Bmatrix}$$ is a finite state.

The input of the automaton are two sequences $u,v:u\in\Sigma^*,v\in\Sigma^*$.

The output is the score of the best alignment of two sequences of amino acids, which can be interpreted as the measure of similarity of the input sequences. The higher score, the more similar the sequences are. As a result, the automaton can be used to compute the similarity function for two sequences of amino acids.

4 Clustering of amino acids sequences

The aim of the clustering the amino acids is to divide the set of sequences into clusters with similar sequences. Each sequence is described only by its constituent amino acids. The dissimilarity function for clustering should be of the form:

A dissimilarity function on a set of sequences X is a function $d:X\times X\to[0,\infty)$ satisfying:

$d(u,u)=0$, $d(u,v)\ge 0$

$d(u,v)=d(v,u)$ (symmetry)

$d(u,z)\le d(u,v)+d(v,z)$

for each $u,v,z \in X$.

The similarity function that has been described in the previous section does not satisfy these conditions. First of all, it is a similarity function instead of dissimilarity function. This problem can be easily overcome if the similarity function is normalized, i.e. $0 \le s(u,v) \le 1, \forall u,v \in X$. For such function the dissimilarity function can be derived as follows:

$d(u,v) = 1 - s(u,v)$,

where s is normalized similarity function and d is a dissimilarity function.

Our similarity function can be normalized by dividing the score of the optimal alignment by the score of an alignment of the longer sequence with itself:

$$s'(u,v) = \frac{s(u,v)}{s(z,z)}, \text{ where}$$

$$\left(z = u \Leftrightarrow \left[\left(|u| > |v|\right) \vee \left(|v| = |u| \wedge s(u,u) \ge s(v,v)\right)\right]\right) \vee \left(z = v \Leftrightarrow \left[\left(|v| > |u|\right) \vee \left(|v| = |u| \wedge s(u,u) < s(v,v)\right)\right]\right)$$

Second, as the gap penalty it is not possible to use value -1, because the function cannot assign the negative numbers. The gap penalty should be set to *0*. By these modifications we get the normalized similarity function of two sequences that is easily convertible into dissimilarity function (as shown previously). Since this new function satisfy all conditions on dissimilarity function for clustering, we can use it for the clustering of protein sequences.

Clustering itself can be done by applying either partitioning or hierarchical agglomerative methods. For partitioning methods it is necessary to specify the number of clusters *k*. At the beginning, the algorithms arbitrarily choose *k* sequences as the initial cluster centers (referred to as consensual sequences). Then, in a cycle, they (re)assign all other sequences to the cluster to which the sequence is the most similar (measured as similarity with the the consensual sequence of the cluster) and update the clusters' consensual sequences. The hierarchical agglomerative methods start by placing each sequence in its own cluster and then merge these atomic clusters into larger and larger clusters. Both methods have some advantages and disadvantages. More information on general clustering methods can be found in [3].

5 Conclusions

This work describes the process of clustering protein sequences and the way how the automata theory can be used as a tool for description of the optimal alignment of protein sequences score computation. A modified finite automaton has been proposed for these purposes. The score, which is the output of the automaton, can be interpreted as a measure of similarity of two input sequences processed by the automaton. In the future work we will try to find other theoretical models (e.g. systolic networks) more convenient for describing this task of bioinformatics.

References

1. Christopher Bystroff, David Baker: *Prediction of Local Structure in Proteins using a Library of Sequence-Structure motifs*, Journal of Molecular Biology 1998, Vol. 281, p. 565 - 577
2. Dan K. Krane, Michael L. Raymer: *Fundamental Concepts of Bioinformatics*, ISBN: 0-8053-4633-3, Benjamin Cummings 2003
3. J. Han, M. Kamber: *Data Mining: Concepts and Techniques*, ISBN: 1-55860-489-8, Academic Press 2001