Brno University of Technology

Faculty of Information Technology

Clustering of Protein Sequences

Brno 2005 **Ivana Rudolfová**

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 the dissimilarity function. These functions have to satisfy few conditions. The similarity of two protein sequences can be described by the score of the best alignment of two protein sequences. From this score the dissimilarity function can be derived. In this paper the Needleman and Wunsch algorithm for finding the best alignment is described and the finite automat that can compute the score for the best alignment (without previous knowledge of the alignment) is introduced here.

Keywords: protein sequence, optimal alignment of two protein sequences, finite automata, clustering, similarity function.

This work has been done within the course *Modern Theoretical Computer Science* lectured by doc. RNDr. Alexander Meduna, Csc.

Ivana Rudolfová

Contents

1. Introduction

Prediction of local structure in 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 the 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.
In this work the process of clustering the amino acids sequences is described in the terms of mathematical functions. The similarity of two sequences is calculated as the score of optimal alignment of these sequences. The finite automata, which can be used to found an optimal alignment of two sequences and calculated its score is also introduced here.

2. Cluster analysis

By this analysis we can divide the objects into clusters (classes). The objects are clustered or grouped based on the principle of maximizing the intraclass similarity and minimizing the interclass similarity. That is, clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters. Each cluster that is formed can be viewed as a class of objects, from which rules can be derived. In comparison with Classification the Cluster analysis is an unsupervised learning approach. The classes of objects are not specified at the beginning of the analysis and even the number of the classes need not to be known. By this approach new patterns and groupings can readily be identified.
In the case of amino acid sequences Ch. Bystroff and D. Baker used the cluster analysis to create the groups of similar recurrent amino acid sequence patterns. They got 82 groups and they 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 element. They created a database of these groups and they described there the structural properties of the most favorable conformations (paradigm sequence) and the confidence of this conformation. This database can be used for prediction of secondary structure of protein. When the sequence contain the pattern from the database, than this part of the sequence can fold in the described structure with some confidence (due to the similarity of a pattern and a paradigm sequence in a database)
For the clustering we need some function, which describe the similarity of two objects. This function should return values that can be simply compared. The similarity of two amino acids sequences of the same length can be determined directly by comparing amino acids at each position, using the matrix PAM or

BLOSUM to determine the penalty for non-match. For the sequences of different length it is useful to find the optimal alignment of these sequences before comparing amino acids at each position. The similarity function serves not only for cluster analysis, but also for assessing the confidence in protein structure prediction.

3. Optimal alignment of sequences

An alignment between 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 residues (the match score), the penalty for aligned pairs of nonidentical residues (the mismatch score) and the penalty for the gap (gap penalty). A simple alignment score for a gapped alignment 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 reflect the fact that some substitutions of amino acids are more common than others and the fact, that some substitutions affect the properties of the protein more than others. The properties of 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, which are usually used for aligning protein sequences – PAM matrices and BLOSUM matrices. They were derived by observing substitution rates among the various amino acid residues 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 to find 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 overcame 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 to apply a dynamic programming approach to the problem of sequence alignment [2].

When we aligning 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 match or mismatch bonus of the first symbols from scoring matrix (PAM). 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 for 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 the two sequences *AVPT* and *AILVPT*, where gap penalty is *-1* and the match score for two Alanines is 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.*

An 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 a gap in the sequence along the left axis. A vertical move represents a gap in the sequence along the top axis, and a diagonal move represents an alignment of the amino acids from each sequence. 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).

		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

So the function, which calculates the similarity between the sequences *AVPT* and *AILVPT*, returns the value 13. For clustering of the sequences we only need to know the similarity of two sequences and it is not necessary to care, for which alignment this score was achieved. So we do not need to reconstruct the optimal alignment. To reconstruct the alignment of the sequences from the partial score table, we need to find a path from the lower rightmost entry in the table to the upper leftmost position.

3.2 Finite automata for the optimal alignment

This chapter contains the description of the special type of finite automata, which can compute the score of the optimal alignment between two sequences of amino acids. At first I would like to remind the definition of the deterministic finite automata.

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

- Q is the finite set of the states
- Σ is a finite input alphabet
- δ is a transition function of the form $\delta : Q \times \Sigma \to 2^Q$
- $g_0 \in Q$ is an initial state
- $F \subseteq Q$ is a set of finite states.

If δ is of the form $\delta : Q \times \Sigma \to Q \cup \{undef\}$, ie. $|\delta(g, a)| = 1,\ \forall g \in Q$, then M is a deterministic finite automata.

Originally finite automat is machine, which sequentially reads the symbols from input tape and it changes its state according to the input symbol (transition function). It is usually used for determining if a string belongs to the specified language. The finite automat accepts the string, if it reaches the finite state after reading all symbols from input tape. All strings accepted by the specified finite automata create the language, which is described by this automat.
To use the finite automata for computing the score of the best alignment of two sequences of amino acids we have to slightly modify them. Our automata will compute the score by using the previously described algorithm for finding the best alignment of two sequences. Here is the list of necessary modifications in automat behavior:
1. This automat should make the transitions in parallel, since we are not able to say, which alignment will be the best one, at the beginning of the simulation.
2. The transition function will not depend on the input symbol. At each state there are three possibilities, what can be done – insertion of gap in first or the second sequence or alignment of two symbols. Therefore form each state there are three transitions, which should be done in parallel. The input symbol is than used for computing the partial score.
3. 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 automata 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.
4. The automat should work with two strings (two sequences of amino acids), so the automat should work with two input tapes.
How the automata should look like accordingly to the previous modifications is shown in figure 4.

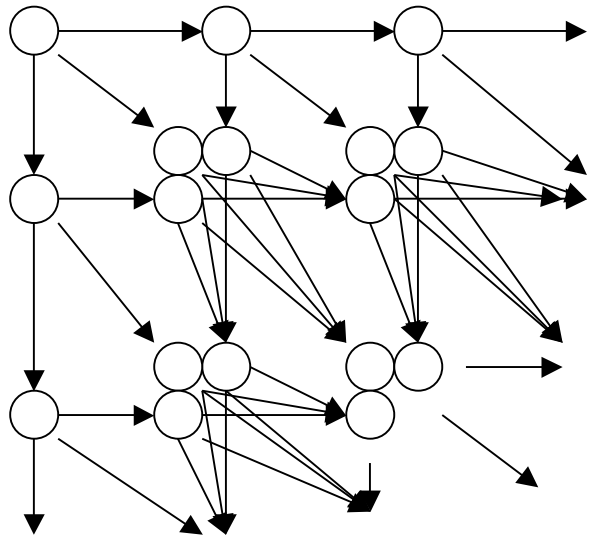

Figure 4: Finite state automata for computing the score of the best alignment of two sequences of amino acids.

The states in this automat we can denote by labels, which corresponds to the positions in the partial scoring tables and to the direction, from which the state was reached – eg. (2,2,v) - the state reached from the state (1,2,h) by vertical transition, (2,2,h) – the state reached from the state (2,1,v) by horizontal transition, (2,2,d) – 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 these automata: horizontal, vertical and diagonal. The horizontal transitions corresponds to reading the symbol from the first sequence (inserting the gap in the second sequence), the vertical transitions corresponds to reading the symbol from the second sequence (inserting the gap in the first sequence) and the diagonal transitions corresponds to reading the symbols from both sequences (aligning of two symbols). Each state contains the value of the partial scores. This score is computed from the partial score from the previous state by adding 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 specified positions are not performed so the value for the next states 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 automata 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 automata is 5-tuple $M = (Q, \Sigma, \delta, g_0, F)$, where

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

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

each state has a value c, $c(1,1) = 0$, $c(x+1,y,h) = \max\{c(x,y,h),(x,y,v),(x,y,d)\} - gp$, $c(x,y+1,v) = \max\{c(x,y,h),(x,y,v),(x,y,d)\} - gp$ and $c(x+1,y+1,d) = \max\{c(x,y,h),(x,y,v),(x,y,d)\} + M(a,b)$, where gp is a gap penalty, a and b are the symbols from input strings (sequences) and $M(a,b)$ is an entry with indexes a and b in used scoring matrix (PAM 250)

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

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

The input of this automat are two strings $u, v : u \in \Sigma^*, v \in \Sigma^*$.
The output is the score of the best alignment of two sequences of amino acids. Hence this values describe the similarity between two sequences (the higher score – the more similar sequences), this automat 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. Therefore it is necessary to find the function, by which we can evaluate the similarity (dissimilarity) of two sequences. The dissimilarity function for clustering should be of the form:

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

$d(x,x) = 0$
$d(x_1,x_2) = d(x_2,x_1)$ (symmetry)
$d(x_1,x_3) \le d(x_1,x_2) + d(x_2,x_3)$
for each $x_1, x_2, x_3 \in X$.

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, when the similarity function is normalized, i.e. $0 \le s(x_1,x_2) \le 1, \forall x_1, x_2 \in X$. For such function the dissimilarity function can be derived as follows:
$d = 1 - s$,
where s is normalized similarity function and d is a dissimilarity function.

To meet this conditions our similarity function can be modified by dividing the score of the optimal alignment by the score of an alignment of the longer sequence with itself:

$$s'(x_1, x_2) = \frac{s(x_1, x_2)}{s(x, x)},$$

$$(x = x_1 \Leftrightarrow [(|x_1| > |x_2|) \vee (|x_2| = |x_1| \wedge s(x_1, x_1) \geq s(x_2, x_2))]) \vee$$

$$(x = x_2 \Leftrightarrow [(|x_2| > |x_1|) \vee (|x_2| = |x_1| \wedge s(x_1, x_1) < s(x_2, x_2))])$$

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.

For the clustering we can use either the partitioning methods or hierarchical agglomerative methods. For partitioning methods it is necessary to specify the number of clusters k. The algorithms arbitrarily choose *k* objects as the initial cluster centers and then in a cycle (re)assign each object to the cluster to which the object is the most similar (similarity with the consensual sequence of each cluster) and update the cluster means (i.e. to find the mean value of the object for each cluster – consensual sequence). The hierarchical agglomerative methods start by placing each object in its own cluster and then merge these atomic clusters into larger and larger cluster. Both methods have some advantages and disadvantages that can be found in a literature [3].

5. Conclusions

This work describes the process of clustering protein sequences. The most common methods for clustering divide the objects on the basis of their similarity. They use some dissimilarity function, which describes the dissimilarity of two objects by some values, which are easily comparable. The main part of this work is devoted to the similarity function for two protein sequences. We use the score of the best alignment of two protein sequences as the similarity function. The way, how to create the dissimilarity function from this score is described in chapter 4. The algorithm how to find the best alignment of two sequences and how to compute the score of this alignment is described in chapter 3.1. This work also introduces a specialized type of finite automata that compute the score of the best alignment of two sequences.

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