Algorithms for Molecular Biology **Fall Semester, 2001**

Lecture 3: December 29, 2001

Lecturer: Ron Shamir *Scribe: Nelly Bluvshtein and Jenny Sannikov* [1]

3.1 Introduction - Sequence Alignment Heuristics

In the second lecture we presented dynamic programming algorithms to calculate the best alignment of two strings. When we are searching a database of size $10^9 - 10^{10}$ for the closest match to a query string of length 200-500, we cannot use these algorithms because they require too much time. There are several approaches to overcome this problem:

1. Implementing the dynamic programming algorithms in hardware, thus executing them much faster. The major disadvantage of this method is its high cost and so it is not available to most researchers.

2. Using parallel hardware, the problem can be distributed efficiently to a number (thousands) of processors, and the results can be integrated later. Like the previous method, this approach is very expensive.

3. Using different heuristics that work much faster than the original dynamic programming algorithm.

A *heuristic method* is an algorithm that gives only approximate solution to a given problem. Sometimes we are not able to formally prove that this solution actually solves the problem, but heuristic methods are commonly used because they are much faster than exact algorithms. In addition, this is a software based strategy, which is therefore relatively cheap and available to any researcher.

In this lecture we present some of the most commonly used heuristics. They are based on the following observations:

1. Even linear time complexity will be problematic when database size is huge (over 10^9).

2. Preprocessing of the database is desirable, since numerous queries are run on an unfrequently updated database.

3. Substitutions are much more likely than indels.

[1]Based in part on scribe by Alexander Shevchenko and Jakov Kostjukovsky, 2000.

4. We expect homologous sequences to contain a lot of segments with matches or substitutions, but without indels and gaps. These segments can be used as starting points for further searching.

3.2 FASTA

The FASTA algorithm is a heuristic method for string comparison. It was developed by *Lipman* and *Pearson* in 1985 [5] and further improved in 1988 [6].

FASTA compares a query string against a single text string. When searching the whole database for matches to a given query, we compare the query using the FASTA algorithm to every string in the database.

Good local alignment is likely to have exact matching subsequences. The algorithm uses this property and focuses on segments in which there will be an absolute identity between the two compared strings. We can use the alignment Dot-Plot matrix (see Figure 3.1) for finding these identical regions.

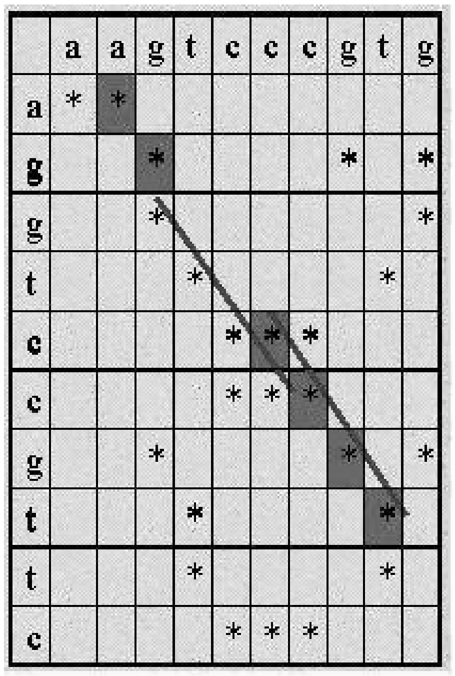


Figure 3.1: Alignment Dot-Plot Matrix.

We next present some definitions concerning FASTA:

- *ktup* (short for k respective tuples) - an integer parameter, which specifies the length of the matching substrings. The standard recommended *ktup* values are 4-6 for DNA sequences matching and 1-2 for protein sequence matching. The trade-off between speed and sensitivity is controlled by this parameter.
- *hot spots* - the matching *ktup*-length substrings. Consecutive *hot spots* are located along the alignment Dot-Plot matrix or dynamic programming matrix diagonals.
- *diagonal run* - a sequence of nearby *hot spots* on the same diagonal (not necessarily adjacent along the diagonal, i.e., spaces between these *hot spots* are allowed).
- $init_1$ - the best scoring run.
- $init_n$ - the best local alignment found by combining "good" diagonal runs with indels in between.

The stages in the FASTA algorithm are as follows:

1. Look for *hot spot*s.

   This stage can be done efficiently by using a lookup table or a hash. For example, we can preprocess the database and store for each possible *ktup* (AA - 20^2, DNA -4^6) exactly where it appears along the database sequence . Then, we can scan the query by shifting a *ktup*-long window and access each *ktup* in the hash for retrieving locations in the database sequence.

2. Find 10 best *diagonal run*s.

   In order to evaluate the diagonal runs, FASTA gives each hot spot a positive score, and the space between consecutive hot spots in a run is given a negative score that decreases with distance. The score of the diagonal run is the sum of the *hot spots* scores and the interspot scores. FASTA finds the 10 highest scoring diagonal runs under this evaluating scheme.

3. Compute $init_1$ and *runs'* filtration.

   A diagonal run specifies an alignment, which is composed of matches (the *hot spots*) and mismatches (from the interspot regions), but does not contain any indels because it is derived from a single diagonal. We next evaluate the runs using an amino acid (or nucleotide) substitution matrix, and pick the best scoring run - $init_1$. In addition, we discard diagonal runs achieving relatively low scores.

4. Combine close *diagonal run*s and compute $init_n$.

   Until now we essentially did not allow any indels in the subalignments. We now try to combine "good" diagonal runs from close diagonals, thus achieving a subalignment with indels. We take "good" subalignments from the previous stage (subalignments whose score is above some specified threshold) and attempt to combine them into a single larger high-scoring alignment that allows some spaces. This is done in the following way:

We construct a directed weighted graph (see Figure 3.2) whose vertices are the subalignments found in the previous stage, and the weight of each vertex is the score (computed in the previous stage) of the subalignment it represents. Next, we extend an edge from subalignment u to subalignment v if v starts at a higher row and column than those at which u ends. We give the edge a negative weight which depends on the number of gaps that would be created by aligning according to subalignment u followed by subalignment v. We can discard long and pass-off edges. Essentially, FASTA then finds a maximum weight path in this acyclic graph. The selected alignment specifies a single local alignment between the two strings - $init_n$. As in the previous stage, we discard alignments with relatively low score.

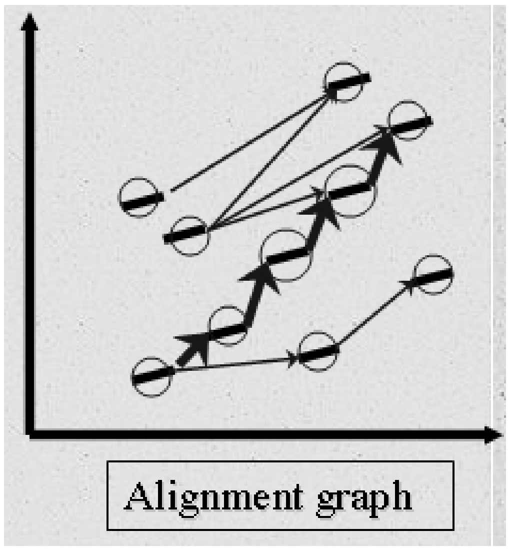


Figure 3.2: The alignment graph.

5. In this step FASTA computes an alternative local alignment score, in addition to $init_n$. Recall that $init_1$ defines a diagonal segment in the dynamic programming matrix. We consider a narrow diagonal band in the matrix, centered along this segment. We observe that it is highly likely that the best local alignment that includes the $init_1$ substrings, lies within the band. We assume this is the case and compute the optimal local alignment in this band, using the ordinary dynamic programming algorithm constrained to that band. Assuming that the best local alignment is indeed within the defined band, the local alignment algorithm essentially merges diagonal runs found in

the previous stages to achieve a local alignment which may contain indels. The band width is dependent on the *ktup* choice. The best local alignment computed in this stage is called *opt*.

6. In the last stage, the database sequences are ranked according to $init_n$ scores or *opt* scores, and the full dynamic programming algorithm is used to align the query sequence against each of the highest ranking result sequences.

Figure 3.3 summarizes the FASTA algorithm stages.

Common FASTA output is a histogram indicating the number of database sequences at any given initial score (see Figure 3.4).

Although FASTA is a heuristic, and as such, it is possible to show instances in which the alignments found by the algorithm are not optimal, it is claimed (and supported by experience) that the resulting alignment scores compare well to the optimal alignment, while the FASTA algorithm is much faster than the ordinary dynamic programming algorithm for sequence alignment.

3.3 BLAST - Basic Local Alignment Search Tool

The *BLAST* algorithm was developed by *Altschul*, *Gish*, *Miller*, *Myers* and *Lipman* in 1990 [1]. The motivation for the development of BLAST was the need to increase the speed of FASTA by finding fewer and better *hot spots* during the algorithm. The idea was to integrate the substitution matrix in the first stage of finding the *hot spots*. The *BLAST* algorithm was developed for protein alignments in comparison to *FASTA*, which was developed for DNA sequences.

BLAST concentrates on finding regions of high local similarity in alignments without gaps, evaluated by an alphabet-weight scoring matrix. Before explaining how *BLAST* obtains its results, we will briefly introduce some terminology.

Given two strings S_1 and S_2, a *segment pair* is a pair of equal length substrings of S_1 and S_2, aligned without gaps. A *locally maximal segment* is a segment whose alignment score (without gaps) cannot be improved by extending it or shortening it. A *maximum segment pair (MSP)* in S_1 and S_2 is a segment pair with the maximum score over all segment pairs in S_1, S_2.

When comparing all the sequences in the database against the query, *BLAST* attempts to find all the database sequences that when paired with the query contain an *MSP* above some cutoff score S. We call such a pair, a hi-scoring pair (*HSP*). We choose S such that it is unlikely to find a random sequence in the database that achieves a score higher than S when compared with the query sequence.

The stages in the BLAST algorithm are as follows:

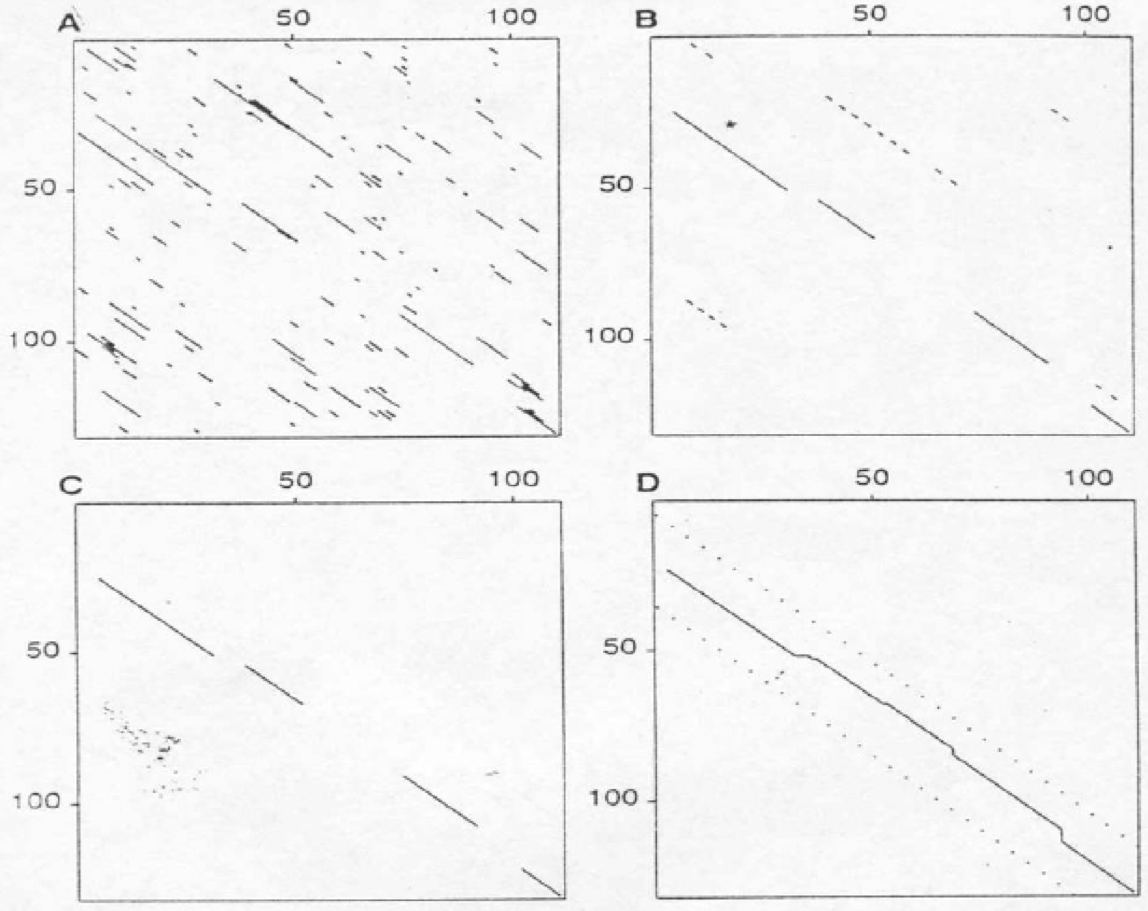
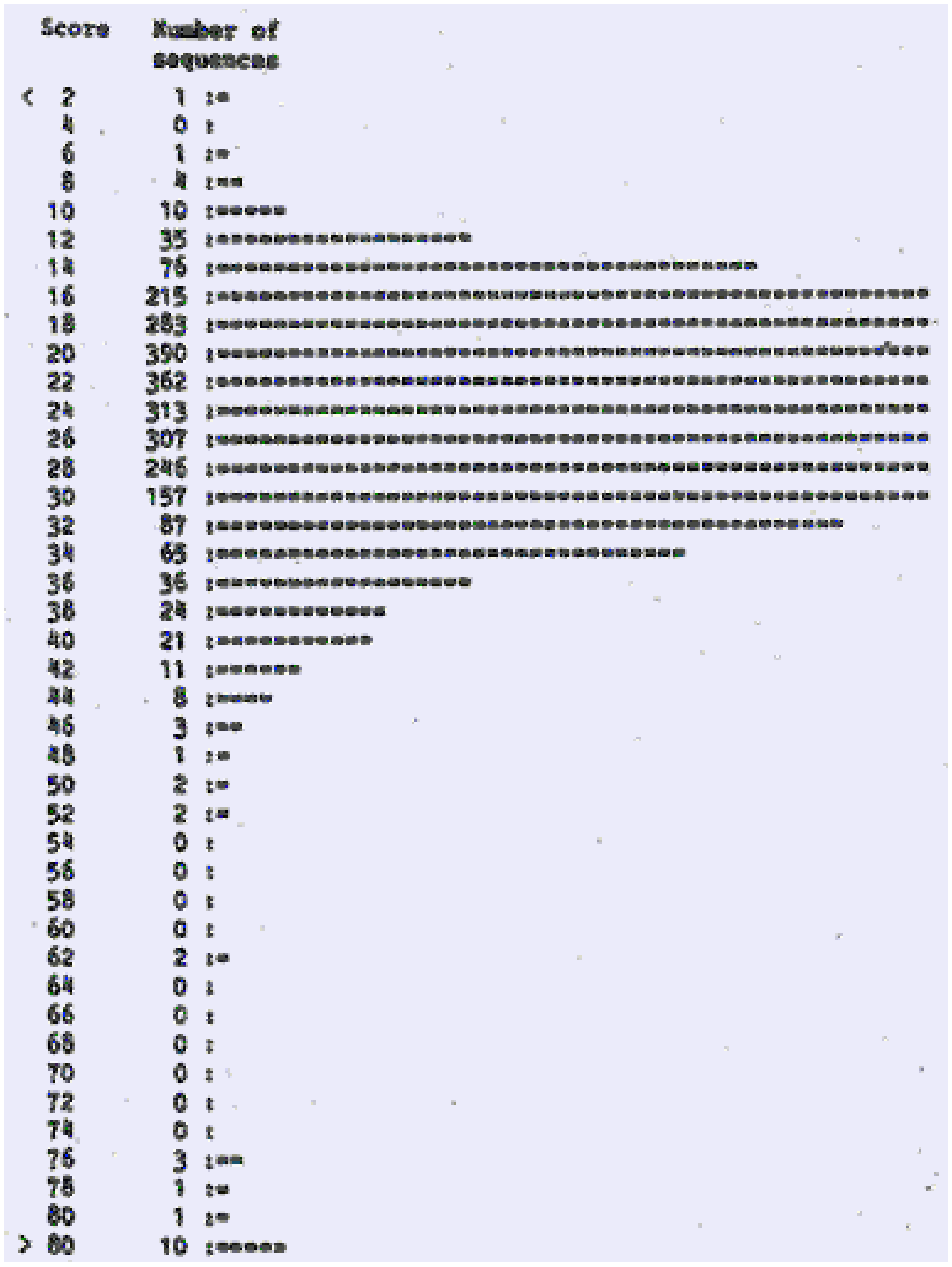


FIG. 1. Identification of sequence similarities by FASTA. The four steps used by the FASTA program to calculate the initial and optimal similarity scores between two sequences are shown. (*A*) Identify regions of identity. (*B*) Scan the regions using a scoring matrix and save the best initial regions. Initial regions with scores less than the joining threshold (27) are dashed. The asterisk denotes the highest scoring region reported by FASTP. (*C*) Optimally join initial regions with scores greater than a threshold. The solid lines denote regions that are joined to make up the optimized initial score. (*D*) Recalculate an optimized alignment centered around the highest scoring initial region. The dotted lines denote the bounds of the optimized alignment. The result of this alignment is reported as the optimized score.

Figure 3.3: Stages of the FASTA algorithm [6].



```
 Score   Number of
         sequences
<  2          1 :=
   4          0 :
   6          1 :=
   8          4 :==
  10         10 :=====
  12         35 :=================
  14         76 :===================================
  16        215 :==============================================
  18        283 :==============================================
  20        390 :==============================================
  22        362 :==============================================
  24        313 :==============================================
  26        307 :==============================================
  28        246 :==============================================
  30        157 :==============================================
  32         87 :=========================================
  34         65 :===============================
  36         36 :=================
  38         24 :===========
  40         21 :==========
  42         11 :======
  44          8 :====
  46          3 :==
  48          1 :=
  50          2 :=
  52          2 :=
  54          0 :
  56          0 :
  58          0 :
  60          0 :
  62          2 :=
  64          0 :
  66          0 :
  68          0 :
  70          0 :
  72          0 :
  74          0 :
  76          3 :==
  78          1 :=
  80          1 :=
> 80         10 :=====
```

Figure 3.4: FASTA output histogram [5].

1. Given a length parameter w and a threshold parameter t, *BLAST* finds all the w-length substrings (called words) of database sequences that align with words from the query with an alignment score higher than t. Each such hot spot is called a *hit* in *BLAST*. Instead of requiring words to match exactly, *BLAST* declares that a word hit has been made if the word taken from the the database has a score of at least t when a substitution matrix is used to compare the word from the query. This strategy allows the word size (w) to be kept high (for speed), without sacrificing sensitivity. It is usually recommended to set the parameter w to values of 3 to 5 for amino acids, and ~ 12 for nucleotides. Thus, t becomes the critical parameter determining speed and sensitivity, and w is rarely varied. If the value of t is increased; the number of background word hits will go down and the program will run faster. Reducing t allows more distant relationships to be found.

2. Extend each *hit* to a *locally maximal segment* and check if its score is above S, i.e. if this sequences pair is *HSP*. Since pair score matrices typically include negative values, extension of the initial w-mer *hit* may increase or decrease the score. Accordingly, the extension of a *hit* can be terminated when the reduction in score (relative to the maximum value encountered) exceeds certain score drop-off threshold.

We may implement the first stage by constructing, for each w-length word α in the query sequence, all the w-length words whose similarity to α is at least t. We store these words in a data structure which is later accessed while checking the database sequences.

Although BLAST does not allow alignments with indels, it has been shown that with the correct selection of values to the parameters used by the algorithm, it is possible to obtain most of the correct alignments while saving much of the computation time compared to the standard dynamic programming method.

3.3.1 Improved BLAST

Altschul et al. suggested in 1997 [3] an improved BLAST algorithm that allows indels in the alignment.

The algorithm stages follow:

1. When considering the dynamic programming matrix to align two strings, we search along each diagonal for two w-length words such that the distance between them is $\leq A$ and their score is $\geq T$. T can be lower than in the previous algorithm. Future expansion is done only to such pairs of hits.

2. In the second stage we want to allow local alignments with indels, similarly to the FASTA algorithm. We allow two local alignments from different diagonals to merge

into a new local alignment composed of the first local alignment followed by some indels and then the second local alignment. This local alignment is essentially a path in the dynamic programming matrix, composed of two diagonal sections and a path connecting them which may contain gaps. Unlike in FASTA, where we only allowed the diagonal to have local shifts, restricted to a band, here we allow local alignments from different diagonals to merge as long as the resulting alignment has a score above some threshold. This method results in an alignment structure which is much less regular.

The improved version of BLAST is about 3 times faster than the original algorithm due to much less expansions made (only two-hit words are expanded).

3.3.2 PSI BLAST - Position Specific Iterated BLAST

The PSI BLAST is another improved version of the BLAST algorithm [3]. When aligning a group of amino acid sequences, i.e., writing them one below the other (see Section 3.5 for discussion of multiple alignment), the vector of characters in a certain column (i.e. the same position in the aligned sequences) is called a *profile*. For a certain profile we may compute the histogram of characters types and obtain their distribution. When we align together amino acid sequences belonging to the same protein family, we will find that some regions are very similar, with profiles showing little variance. These regions, called *conserved regions*, define the structure and functionality typical to this family. We would like the substitution matrices we use to take into account the statistical information that we have regarding how conserved is the column, in order to improve our alignment score.

In the first stage of the algorithm we perform ordinary BLAST while using a different cost vector V_i for each column i. Initially, each such vector V_i is set to the row of the substitution matrix corresponding to the i-th character in the query sequence. From the high-scoring results we get, we build profiles for each column. We continue to perform BLAST iteratively while using as query the collection of profiles, i.e. we use a histogram at each column rather than a simple string, and compare it against the database. This is equivalent to updating the position dependent cost vectors according to the profile statistics. After each iterative step we update the profiles according to the obtained result sequences. We terminate the iterative loop when we no longer find new meaningful matches.

PSI-BLAST is a rather permissive alignment tool, finding more distantly related sequences than FASTA or BLAST. It should be used with care, as the studied sequences may prove too distant to be meaningfully related.

3.4 Amino Acids Substitution Matrices

When we search for the best alignment between two protein sequences, the scoring (or substitution) matrix we use can largely affect the results we get. Ideally, the scores should reflect the underlying biological phenomena that the alignment seeks to expose. For example - in the case of sequence divergence due to evolutionary mutations, the values in the scoring matrix should ideally be derived from empirical observation on ancestral sequences and their contemporary descendants.

Two examples of simple substitution matrices often used that do not employ the biological phenomena are:

1. The *unit matrix*:

$$M_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

2. The *genetic code matrix.* M_{ij} equals the number of minimal base substitutions needed to convert a codon of amino acid i to a codon of amino acid j. We disregard here the importance of chemical properties of the amino acids, that evidently influence the chances for their substitution, like their hydrophobicity, charge or size.

Note that (1) measures similarity and (2) measures distance.

3.4.1 PAM units and PAM matrices

The methodology of PAM units and the first specific PAM matrices were developed by Margaret Dayhoff et al. [7, 2]. Dayhoff and her coworkers examined 1572 accepted mutation between 71 superfamilies of closely related sequences of proteins. During this process they noticed that the substitutions that occurred in families of closely related proteins were not random. They concluded that some amino acid substitutions occurred more readily than others, probably because they did not have a great effect on the structure and function of a protein. This meant that evolutionarily related proteins did not need to have the same amino acids at every position: They could have comparable ones. In doing alignments, this becomes very important. From such observations, the PAM matrix was born. PAM stands for "Point Accepted Mutations" or "Percent of Accepted Mutations" [9].

PAM units

We use PAM units to measure the amount of evolutionary distance between two amino acid sequences. Two sequences S_1 and S_2 are at evolutionary distance of one PAM, if S_1 has converted to S_2 with an average of one accepted point-mutation event per 100 amino acids. The term "accepted" here means a mutation that was incorporated into the protein and passed to its progeny. Therefore, either the mutation did not change the function of the protein or the change in the protein was beneficial to the organism.
Note that two strings which are one PAM unit diverged do not necessarily differ in one percent, as often mistakenly thought, because a single position may undergo more than a single mutation. The difference between the two notions grows as the number of units does.
There are two main problems with the notion of PAM units:

1. Practically all the sequences we can obtain today are extracted from extant organisms. We almost do not know any protein sequences where one is actually derived from the other. The lack of ancestral protein sequences is handled by assuming that amino acid mutations are reversible and equally likely in either direction. This assumption, together with the additivity property of the PAM units derived from its definition, imply that given two amino acid sequences S_i and S_j whose mutual ancestor is S_{ij} we have:
$$d(S_i, S_j) = d(S_i, S_{ij}) + d(S_{ij}, S_j)$$
where $d(i, j)$ is the PAM distance between amino acid sequences i and j.

2. Insertions and deletions which may occur during evolution are ignored. Hence we cannot be sure of the correct correspondence between sequence positions. In order to know the exact correspondence one has to be able to identify the true historical gaps, or at least to identify large intervals along the two sequences where the correspondence is correct. This cannot always be done with certainty, especially when the two sequences are evolutionarily distant.

PAM matrices

PAM matrices are amino acid substitution matrices that encode the expected evolutionary change at the amino acid level. Each PAM matrix is designed to compare two sequences which are a specific number of PAM units apart. For example - the PAM120 score matrix is designed to compare between sequences that are 120 PAM units apart: The score it gives a pair of sequences is the (log of the) probabilities of such sequences evolving during 120 PAM units of evolution. For any specific pair (A_i, A_j) of amino acids the (i, j) entry in the PAM-N matrix reflects the frequency at which A_i is expected to replace A_j in two sequences that are n PAM units diverged. These frequencies should be estimated by gathering statistics on replaced amino acids.

How the PAM matrix was generated

Collecting statistics about amino acids substitution in order to compute the PAM matrices is relatively difficult for sequences that are evolutionarily distant, as mentioned in the previous section. Luckily, for sequences that are highly similar, i.e., the PAM divergence distance between them is small, finding the position correspondence is relatively easy since only few insertions and deletions took place.

Therefore, Dayhoff started with aligned sequences, highly similar with known evolutionary trees (71 trees were examined). Then statistics on exchanges were collected (1572 exchanges in total). The PAM1 matrix was computed as follows: Let M_{ij} denote the observed frequency (= estimated probability) of amino acid A_i mutating into amino acid A_j during one PAM unit of evolutionary change. The resulting matrix M is a 20×20 real matrix, with the values in each matrix column adding up to 1. There is a significant variance between the values in each column. For example, see figure 3.5, taken from [2].

	A	R	N	D	C
A	9867	2	9	10	3
R	1	9913	1	0	1
N	4	1	9822	36	0
D	6	0	42	9859	0
C	1	1	0	0	9973

Figure 3.5: The top left corner 5×5 of the M matrix used to compute PAM1. We write $10^4 M_{ij}$ for convenience.

Note that a matrix with an evolutionary distance of 0 PAMs would have ones on the main diagonal and zeros elsewhere. A matrix with an evolutionary distance of 1 PAM would have numbers close to one on the main diagonal and small numbers off the main diagonal. One PAM would correspond to roughly 1% divergence in a protein (one amino acid replacement per hundred). To derive a mutational probability matrix for a protein sequence that underwent N percent accepted mutations, we use the PAM-N matrix, the PAM1 matrix multiplied by itself N times. This results in a family of scoring matrices, each suitable for a given evolutionary distance.

Once M is known, the matrix M^n gives the probabilities of any amino acid mutating to any other during n PAM units. For convenience the following matrix is derived from PAM-N:

$$c_{ij} = \log \frac{f(j)M^n(i,j)}{f(i)f(j)} = \log \frac{M^n(i,j)}{f(i)}$$

where $f(i)$ and $f(j)$ are the observed frequencies of amino acids A_i and A_j respectively. This approach assumes that the frequencies of the amino acids remain constant over time, and that the mutational processes causing substitutions during an interval of one PAM unit operate in the same manner for longer periods (assuming constant evolutionary clock). In general, the key idea here is to use reliable information from similar sequences (getting PAM1) and then extrapolating it. We are using "log odds" - taking the log value of the probability in order to allow computing the total score of all substitutions using summation rather than multiplication. Partial validation of the correctness of PAM1 can be gained by viewing the PAM matrices organized by groups of similar amino acids, when all group members are located in consecutive columns in the matrix.

3.4.2 BLOSUM - BLOcks SUbstitution Matrix

The BLOSUM matrix is another amino acid substitution matrix, first calculated by *Henikoff* and *Henikoff* [4]. The difference between the PAM and BLOSUM matrices is that PAM is derived from global alignments of proteins, while BLOSUM comes from alignments of shorter sequences – blocks of sequences that match each other at some defined level of similarity. The BLOSUM method thereby incorporates much more data into its matrices, and is therefore, presumably, more accurate.

For BLOSUM matrix calculation, only blocks of similar amino acid sequences are considered. We define blocks as a conserved region of a protein family with the family members aligned (see Figure 3.6). These blocks represent over 500 groups of related proteins, and act as signatures of these protein families. One reason for this is that one needs to find a multiple alignment between all these sequences and it is easier to construct such an alignment with similar sequences. Another reason is that the purpose of the matrix is to measure the probability of one amino acid changing into another, and the change between distant sequences may include also insertions and deletions of amino acids. Moreover, we are more interested in conservation of regions inside protein families, where sequences are quite similar, and therefore we restrict our examination to such.

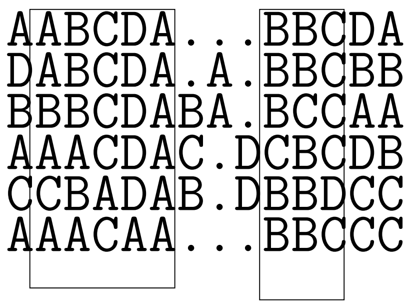


Figure 3.6: Alignment of several sequences. The conserved blocks are marked.

The first stage of building the BLOSUM matrix is eliminating sequences, which are identical in more than $x\%$ of their amino acid sequence. This is done to avoid bias of the result in favor of a certain protein. The elimination is done either by removing sequences from the block, or by finding a cluster of similar sequences and replacing it by a new sequence that represents the cluster. If two sequences are more than $x\%$ identical, then the contribution of these sequences is weighted to sum to one. In this way the contributions of multiple entries of closely related sequences is reduced. The matrix built from blocks with no more the $x\%$ of similarity is called BLOSUM X (e.g., the matrix built using sequences with no more then 50% similarity is called BLOSUM50.)

Secondly, substitution statistics in each column of each block are collected. This is done by counting the pairs of amino acids in each column of the multiple alignment. For example in a column with amino-acids AABACA (as in the first column in the block in Figure 3.6), there are 6 AA pairs, 4 AB pairs, 4 AC, one BC, 0 BB, 0 CC. So we have a contribution of 6+4+4+1=15 from the example column to the count (more generally, $\binom{n}{2}$ pairs for a column of n amino-acids).

In the last stage the results are normalized according to the following definitions:

1. $q_{i,j}$ - an observed probability for a pair of amino acids in the same column to be A_i and A_j.

2. p_i - an observed probability of a certain amino acid to be A_i:

$$p_i = q_{i,i} + \sum_{i \neq j} q_{i,j}/2$$

3. Assuming independence, given pairs should occur with frequencies $e_{i,j}$ as follows:

$$e_{i,j} = 2p_ip_j, \text{ if} i \neq j$$

4. The odds matrix is $q_{i,j}/e_{i,j}$.

5. *log odd ratio* is calculated as $s_{i,j} = \log_2 \frac{q_{i,j}}{p_ip_j}$. The final result is the rounded $2s_{i,j}$. This value is stored in the (i,j) entry of the BLOSUM matrix. If the observed number of differences between a pair of amino acids is equal to the expected number then $s_{i,j} = 0$. If the observed is less than expected then $s_{i,j} < 0$ and if the observed is greater than expected then $s_{i,j} > 0$.

How can we compare the performance of a BLOSUM matrix with a PAM matrix? Which clustering percentage matrix from the BLOSUM family is equivalent with what PAM distance? Some tests were performed [10] and the results were that BLOSUM 45, 62, and 80 performed better (missed less aligned positions) than PAM 120, 160 or 250. While the PAMs missed 30-31 positions, BLOSUM missed 6-9 positions. In contrast to the PAM matrices, more sequences are examined in the process of computing the BLOSUM matrix. Moreover, the sequences are of specific nature of resemblance, and therefore the two sets of matrices differ.

Comparing the efficiency of two matrices is done by calculating the ratio between the number of pairs of similar sequences discovered by a certain matrix but not discovered by another one and the number of pairs missed by the first but found by the other. According to this comparison BLOSUM62 (see example [8]) is found to be better than other BLOSUM X matrices as well as PAM-X matrices and highly recommended for sequence alignment and database searching. In general, it is logically to use a specific matricies for a certain protein.

Bibliography

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215:403–10, 1990.

[2] M. Dayhoff and R. Schwartz. Matrices for detecting distant relationship. *Atlas of Protein Sequences*, pages 353–358, 1979.

[3] S. F. Altschul et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, 25(17):3389–402, 1997.

[4] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Science USA*, 89(22):10915–10919, November 1992.

[5] D. Lipman and W. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

[6] D. Lipman and W. Pearson. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Science USA*, 85:2444–2448, 1988.

[7] R. Schwartz M. Dayhoff and B. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:345–352, 1978.

[8] `http://helix.biology.mcmaster.ca/721/distance/node10.html`.

[9] `http://helix.biology.mcmaster.ca/721/distance/node9.html`.

[10] `http://www.ncgr.org/bioinformatics/gbs/VSNS/Blosum.html`.