CPS260/BGT204.1 Algorithms in Computational Biology **September 30, 2003**

Lecture 10: Local Alignment and Substitution Matrices

Lecturer: Pankaj K. Agarwal *Scribe: Madhuwanti Vaidya*

So far we have seen global alignment, where entire sequences are matched. There are two other variations of global alignment.

10.1 Semiglobal alignment

In semiglobal alignment we do not pay penalty for end gaps. These are gaps that appear before the first letter of the sequence or after the last letter of the sequence. They are also called leading and trailing gaps, respectively. If one of the sequences is significantly shorter than the other, then semiglobal alignment is preferrable.

Example
Consider two sequences -
C A G C A C T T G G A T T C T C G G and C A G C G T G G.
They can be aligned in many ways:

Alignment 1 Alignment 2

Figure 10.1: Semi-global alignment

We want to choose the Alignment1 over Alignment2 as Alignment2 fragments the second sequence, which is not what we are looking for. Giving Alignment1 a better score as compared to Alignment2 is done by not paying a penalty for the trailing and leading gaps.

Trailing Gaps Suppose we have two sequences $U = \langle u_1, ..., u_n \rangle$, and $V = \langle v_1, ..., v_n \rangle$ and V is the shorter of the 2 sequences. We wish to align V with $u_1, u_2, ..., u_i$ and insert trailing gaps. Instead of looking only at the entry $\sigma(m, n)$ in the matrix computed by the DP algorithm for global alignment, we look at all the entries in last column of the matrix. The i^{th} entry in the last column of the matrix indicates the score of aligning V with $u_1, ..., u_i$. The maximum from this column is chosen and returned as the score of aligning V with U,

ignoring the trailing gaps. In case U is the shorter sequence then look at the last row of the matrix and chose the maximum. That is we return

$$\max_{1 \le i \le m} \sigma(i, n).$$

Leading Gaps If leading gaps are to be ignored, only the initialization routine of the earlier algorithm has to be changed. Thus

$$\sigma(i, 0) = 0 = \sigma(0, j).$$

End Gaps Combining the two methods discussed above, we do not charge any gap penalty for leading and trailing gaps. Hence maximum of the last column (or row) is found and returned as the score of that semi-global alignment.

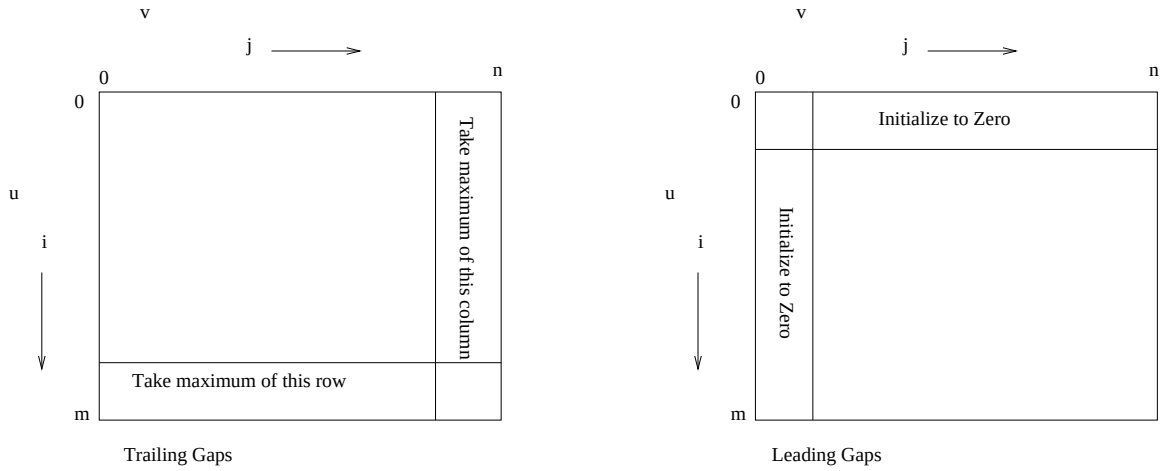

Figure 10.2: Handling end Gaps

10.2 Local Alignment

This is used to find conserved regions in two long sequences. Here only parts of the sequences are aligned. Thus a substring from one sequence is aligned with a substring from the other. This can be done by several methods, but their running time complexities vary greatly.

Basically one has to find two pointers i_1, j_1 in U and i_2, j_2 in V, such that $u_{i_1}, ..., u_{j_1}$ locally match with $v_{i_2}, ..., v_{j_2}$. Let $U_{i_1 j_1} = \langle u_{i_1}, ..., u_{j_1} \rangle$ and $V_{i_2 j_2} = \langle v_{i_2}, ..., v_{j_2} \rangle$. We define the best score of aligning U and V locally as :

$$\lambda(U, V) = \max_{\substack{1 \le i_1 \le j_1 \le m \\ 1 \le i_2 \le j_2 \le n}} \sigma(U_{i_1 j_1}, V_{i_2 j_2})$$

1. Define the function $\rho(i_1, j_1, i_2, j_2) = \sigma(U_{i_1 j_1}, V_{i_2 j_2})$ which gives the score of the alignment of $u_{i_1}, ..., u_{j_1}$ with $v_{i_2}, ..., v_{j_2}$. The maximum over all possible values of i_1, j_1, i_2, j_2 is $\lambda(U, V)$. Assuming $m = n$, this requires O(n^6) time. The number of subproblems are O(n^4), as there n different values for each variable. Also solving each subproblem using DP requires O(n^2) time. Hence the total time required is O(n^6).

2. This can be improved to O(n^4) by a slight modification. Here we fix i_1, i_2 and run the Dynamic Programming global-alignment algorithm, for the sequences $u_{i_1}, ..., u_n$ and $v_{i_2}, ..., v_n$. Then the index of the maximum entry in the DP score table determines j_1, j_2. The next step is to try all possible combinations of i_1, i_2. Thus the running time is reduced to O(n^4), as there are n^2 subproblems and DP takes O(n^2) time.

3. With some change in the original DP the running time can be further reduced to O(n^2). The iterative definition of the score function is:

$$\sigma(i,j) = \max \begin{cases} \sigma(i-1, j-1) + p(i,j) \\ \sigma(i-1, j) + g \\ \sigma(i, j-1) + g \end{cases}$$

   where $p(i, j)$ is as usual the score of matching u_i with v_j.

   We can see that the value of the $(i, j)^{th}$ entry in the matrix depends on $(i-1, j-1)^{th}$, the $(i, j-1)^{th}$ and the $(i-1, j)^{th}$ entries, which are previously computed. If the maximum of these is < 0, then we should start a new alignment from this position. This is because adding a negative value is not going to improve the score, so we could just as well take the score to be 0 and start a new local alignment at the $(i, j)^{th}$ entry. This means that now we will be looking for alignments in the sequences $u_i, ..., u_n$ and $v_j, ..., v_n$. The initialization of the 0^{th} row and column of the matrix also changes due to this. It becomes:

$$\sigma(i, 0) = \sigma(0, j) = 0.$$

   The new iterative function for the score becomes:

$$\sigma(i,j) = \max \begin{cases} 0, \\ \sigma(i-1, j-1) + p(i,j), \\ \sigma(i-1, j) + g, \\ \sigma(i, j-1) + g. \end{cases}$$

   Finally the maximum of all the entries is returned as the score of the best possible local alignment. Thus using dynamic programming the running time for the problem of finding local alignment is reduced to O(n^2).
   This algorithm is known as the *Waterman-Smith Algorithm*.

10.3 Substitution Matrices

The function $p(i, j)$ which is used in computing the best alignment is easy to define if we are aligning DNA sequences. In case of protein matching this function is not so well defined. We shall look at each case separately.

In case of matching DNA sequences, the function is simple. It is either 1 or 0 depending on whether a match occurred or not. If there is a perfect match, which is A aligns with A, T aligns with T, C aligns with C and

G aligns with G, the function returns 1 else it returns 0.

$$p(i,j) = \begin{cases} 1 & \text{if } u_i = v_j, \\ 0 & \text{otherwise.} \end{cases}$$

The substitution matrix for DNA matching, with the alphabet A,T,G and C looks as follows:

	A	T	G	C
A	1	0	0	0
T	0	1	0	0
G	0	0	1	0
C	0	0	0	1

The score matrix becomes more complicated for protein-sequence alignment because the alphabet size increases from 4 to 20(as there are 20 amino acids) and also because the scoring scheme of 1 for a match and 0 for a mismatch is not enough. This is called the *unit matrix* scheme.This measures the similarity between the two strings.

Another scheme called as the *genetic code matrix* where $M_{(i,j)}$ equals the number of minimal base substitutions needed to convert a codon of amino acid i to a codon of amino acid j.This measures the distance between the two amino acid sequences.

In both schemes we disregard the functionality and the chemical properties of the amino acids. Because protein comparisons are often made with evolutionary concerns in mind, the scores should reflect the underlying biological phenomena that the alignment seeks to pose. A certain mutation that changes one amino acid into another may not affect the function of the protein as more than one amino acids could perform the same function. Thus some mutations should be given a better score as compared to the others. The factors that influence the probability of mutual substitution are so numerous and varied that direct observation of actual substitution rates is often the best way of deriving similarity scores. There are a number of ways this can be done. The most popular techniques used are

- BLOSUM
- PAM
- GONNET

We shall discuss the first two here.

BLOSUM This is an acronym for BLOcks SUbstitution Matrices. This method was developed by Henikoff and Henikoff in 1992 [1]. The substitution matrix is obtained by using blocks of similar amino acid sequences as data and then applying statistical methods to the data to obtain the similarity scores.

Initially a simple unit matrix is used, where the score is 1 for a match and 0 for a mismatch. Then, from the sets of proteins from public databases that have been grouped into related families, some blocks of sequences (ungapped sequences) that appear to have been conserved are found . Thus this is not dependent on any specific scoring scheme as the starting point is the unit matrix.

Let us consider a very simple example:

Consider only the alphabet A, B, C and only one block consisting of 6 sequences.

Sequence	Position			
Seq1	B	A	B	A
Seq2	A	A	A	C
Seq3	A	A	C	C
Seq4	A	A	B	A
Seq5	A	A	C	C
Seq6	A	A	B	C

Using this data block, the number of occurrences of each amino acid and also the number of pairs of amino acids aligned in the same column are counted. Here we see that A occurs 14 times, B occurs 4 times and C occurs 6 times out of a total of 24 (14 + 4 + 6). Counting the total number of aligned pairs (vertically - column wise), there are $4.\binom{6}{2} = 60$ aligned pairs. The frequency of the pair AA is the greatest and that of BB the least.

Pair	Frequency of occurrence
AA	26
AB	8
AC	10
BB	3
BC	6
CC	7

Now these observed frequencies are compared to the expected number of times an amino acid pair is aligned if a random collection of amino acids is observed. If two sequences are chosen at random with the probability of occurrence of A as 14/24, B as 4/24 and C as 6/24 and then aligned, then the probability of an A aligning with another A is 14/24 * 14/24. Similarly the probability of a B aligning with an A is 2 * 14/24 * 4/24. (There is a factor of 2 as the alignment could be A from the first sequence and B from the second or it could be B from the first sequence and A from the second.)

Pair	Observed (O)	Expected (E)	$2\log_2(O/E)$
AA	26/60	196/576	0.70
AB	8/60	112/576	-1.09
AC	10/60	168/576	-1.61
BB	3/60	16/576	1.70
BC	6/60	48/576	0.53
CC	7/60	36/576	1.80

The last column gives the ratio of the occurrence each amino acid combination in the observed data to the expected value of occurrence of the pair. This is called the *log odd ratio*. It is rounded off and used in the substitution matrix. The substitution matrix for this data looks as follows:

	A	B	C
A	1	-1	2
B	-1	2	1
C	-2	1	2

This method gives a positive score to those pairs that are more likely than chance to occur and negative scores to those pairs that are less likely.

Table 2 - The log odds matrix for BLOSUM 62

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	4	0	-2	-1	-2	0	-2	-1	-1	-1	-1	-2	-1	-1	-1	1	0	0	-3	-2
C		9	-3	-4	-2	-3	-3	-1	-3	-1	-1	-3	-3	-3	-3	-1	-1	-1	-2	-2
D			6	2	-3	-1	-1	-3	-1	-4	-3	1	-1	0	-2	0	-1	-3	-4	-3
E				5	-3	-2	0	-3	1	-3	-2	0	-1	2	0	0	-1	-2	-3	-2
F					6	-3	-1	0	-3	0	0	-3	-4	-3	-3	-2	-2	-1	1	3
G						6	-2	-4	-2	-4	-3	0	-2	-2	-2	0	-2	-3	-2	-3
H							8	-3	-1	-3	-2	1	-2	0	0	-1	-2	-3	-2	2
I								4	-3	2	1	-3	-3	-3	-3	-2	-1	3	-3	-1
K									5	-2	-1	0	-1	1	2	0	-1	-2	-3	-2
L										4	2	-3	-3	-2	-2	-2	-1	1	-2	-1
M											5	-2	-2	0	-1	-1	-1	1	-1	-1
N												6	-2	0	0	1	0	-3	-4	-2
P													7	-1	-2	-1	-1	-2	-4	-3
Q														5	1	0	-1	-2	-2	-1
R															5	-1	-1	-3	-3	-2
S																4	1	-2	-3	-2
T																	5	0	-2	-2
V																		4	-3	-1
W																			11	2
Y																				7

Figure 10.3: BLOSUM 62 Substitution Matrix

To summarize the method, we see that BLOSUM matrices are based on local alignments. We start with blocks of sequence fragments from different protein families which can be aligned without the introduction of gaps. These sequence blocks correspond to highly conserved regions. Amino acid pair frequencies are then compiled from these blocks simply by summing over all possible pairs of the block. These frequencies are written down in a frequency table, and the odds for relatedness are calculated, which are then rounded off to get the BLOSUM matrix.

One of the shortcomings of this method is that it overlooks an important factor that could bias the results. If there are many closely related proteins in one block and a few others that are less closely related, then the substitution matrix will be biased toward the closely related proteins. To reduce this bias, grouping of closely related proteins in one block is done and their contributions are normalized.

In the first stage of building the BLOSUM matrix, while choosing the blocks, amino acid sequences which are more than $x\%$ identical are grouped or clustered together. Thus the contributions of multiple entries of closely related sequences is reduced, as their contribution is weighted to sum to one. The matrix built from blocks with no more than $x\%$ of similarity is called BLOSUM X, for example if the 0.85 similarity criterion is adopted, the final matrix is called BLOSUM 85.

A further refinement to this is that after obtaining the substitution matrix, it is used instead of the conservative "unit" matrix to construct the set of blocks. This process can be repeated again and it is from this third set of blocks that the final BLOSUM matrices are defined.

PAM This stands for Point Accepted Mutation or Percent of Accepted Mutations. This was introduced by Dayhoff, Schwartz, and Orcutt in 1978 [2].

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.

The basic 1-PAM reflects an amount of evolution producing on an average one mutation per hundred amino acids. Two sequences S1 and S2 are at evolutionary distance of one PAM, if S1 has converted to S2 with an average of one accepted point-mutation event per 100 amino acids. The score given by a 1-PAM to a pair of sequences is the (log of the) probabilities of such sequences evolving during 1-PAM unit of evolution. For any specific pair (A_i, A_j) of amino acids the $(i,j)^{th}$ 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.

PAM matrices are calculated as follows:
To construct PAM matrices, the matrix corresponding to 1 PAM is computed first and the other matrices are calculated from it. Also, mutations are viewed at the amino acid level only, not the DNA level. For each evolutionary distance we have a *probability transition matrix* M and a *score matrix* S. A list of *accepted mutations* and the probabilities of occurrence p_a for each amino acid a is required to compute the entries of M.

An *accepted mutation* is a mutation that occurred and was positively selected by the environment, which means a mutation that was incorporated into the protein and either it did not change the function of the protein or the change was beneficial to the organism. One way to collect accepted mutations is to align two homologous proteins from different species and the positions where the sequences differ give us an accepted mutation. These accepted mutation are treated as undirected events, as we do not know within a pair of amino acids, which amino acid mutated into the other .

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. Hence we should consider immediate mutations like $a \rightarrow b$, not mediated ones like $a \rightarrow c \rightarrow b$. The possibility of occurrence of mediated mutations is minimized by taking very closely related sequences.

The probabilities of occurrence p_a for each amino acid a can be estimated simply by computing the relative

frequency of occurrence of the amino acid over a large, sufficiently varied protein sequence. Then from the list of accepted mutations we compute the number of times the mutation $a \leftrightarrow b$ was observed to occur. This is denoted by f_{ab}, since it is undirected mutations that we deal with $f_{ab} = f_{ba}$. We also require f_a, which is the total number of mutations in which a was involved.

$$f_a = \sum_{a \neq b} f_{ab}.$$

The total number of amino acid occurrences involved in mutations f is given by:

$$f = \sum_a f_a.$$

Thus f is twice the total number of mutations observed, as there are two amino acids involved in each mutation. This is all that is required to calculate the entries of the transition matrix M, where each entry M_{ab} denotes the observed frequency of amino acid a mutating into amino acid b during one PAM unit of evolutionary change.

If a and b are same, this means that we are calculating the probability of a remaining unchanged during this interval. This is based on *relative mutability* of amino acid a , which is defined as:

$$m_a = \frac{f_a}{100 f p_a}.$$

The mutability of an amino acid is a measure of how much it changes in the evolutionary period of interest. Thus M_{aa}, which represents the probability of a remaining unchanged is:

$$M_{aa} = 1 - m_a.$$

If a and b are different then the probability of a changing into b can be computed as the product of the conditional probability that a will change into b, given that a changed, times the probability of a changing. Thus

$$\begin{aligned} M_{ab} &= \Pr(a \to b) \\ &= \Pr(a \to b | a \text{ changed}) * Pr(a \text{ changed}) \\ &= \frac{f_{ab}}{f_a} m_a. \end{aligned}$$

The resulting matrix M is a 20×20 real matrix, with the following properties:

$$\sum_b M_{ab} = 1,$$

$$\sum_a p_a M_{aa} = 0.99.$$

Notice that we are computing these probabilities using a simplified model of protein evolution where an amino acid mutates independently of its past history. The independence from past history leads to a Markov-type model of evolution.

The unit of evolution used is this model is the amount of evolution that will change 1 in 100 amino acids on average. This is referred to as 1 PAM evolutionary distance. The transition probability matrix is normalized to reflect this fact. This is done while calculating the mutability m_a of a the denominator contains a factor of 100. If another number, say 50, is used then the matrix M would still satisfy the same properties except that it would now reflect a 1 in a 50 average change, that means one unit of evolutionary distance would now mutate one in 50 amino acids. Once the basic PAM matrix is derived we can compute transition probabilities for larger amounts of evolution. The M^k matrix represents the transition probability matrix for a period of k units of evolution.

Table 1 - The log odds matrix for 250 PAMs (multiplied by 10)

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

Figure 10.4: PAM 250 Substitution Matrix

Now we can define scoring matrices. The entries in these matrices are related to the ratio between the probability that a pair is mutated as opposed to being a random occurrence. This is the same technique that we used in BLOSUM, where the *log odd ratio* was calculated. Thus the score is calculated as follows:

$$\text{score}(a,b) = 10 \log_{10} \frac{M_{ab}}{p_b}.$$

This is for 1-PAM distance, but we can compute the k PAM distance similarly:

$$\text{score}_k(a,b) = 10 \log_{10} \frac{M^k_{ab}}{p_b}.$$

These matrices are symmetric due to the fact that we look at undirected mutations. Also low PAM numbers

are good for finding short, strong local similarities, while high PAM numbers are used to detect long, weak ones.

The main 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 incorporates much more data into its matrices and is hence supposed to be more reliable.

A BLOSUM 80 matrix is comparable to a 1-PAM matrix. Since BLOSUM 80 would group 80% similar proteins it would correspond to a smaller evolutionary distance between them. While a BLOSUM 45 matrix would be equivalent to PAM 250.

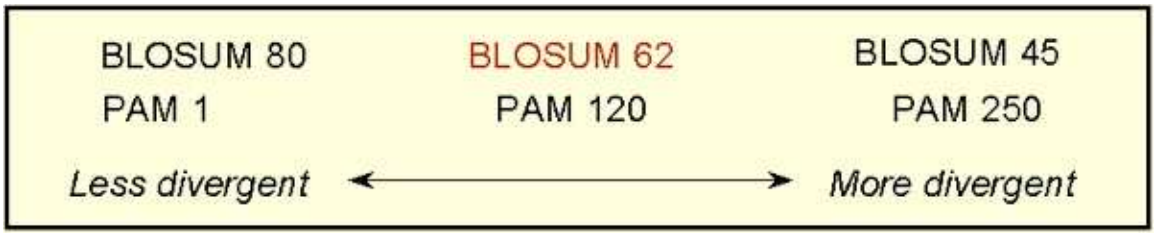

Figure 10.5: BLOSUM and PAM

NCBI recommendations for choosing BLOSUM or PAM are:

- Default matrix to use is BLOSUM62
- For long and distantly related proteins: BLOSUM45
- For closely related sequences: Don't care.
- For short sequences: PAM30 (length < 35), PAM70 (length < 50)

References

[1] Henikoff Steven and Henikoff Jorja. *Amino acid substitution matrices from protein blocks*, PNAS 1992 89: 10915-10919.

[2] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. *A model of evolutionary change in proteins*, National Biomedical Research Foundation, 1978. Atlas of Protein Sequence and Structure, Vol. 5, Suppl. 3, chapter 22, pages 345–352.

[3] W. Evans and G. Grant. *Statistical Methods in Bioinformatics*, Springer-Verlag, 2002.

[4] Mount D.W. *Bioinformatics: Sequence and Genome Analysis*, Cold Spring Harbor Lab Press, 2000.

[5] Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*, Brooks/Cole, Pacific Grove, CA, 1997.