BIOL591: Introduction to Bioinformatics
Scoring and protein alignments

Reading in text (Mount *Bioinformatics*):

There is much in the text on the topics covered below. You'll have to decide for yourself if the explanation helps or hinders understanding.

- pp.300-307 Description of Blast
- pp. 76-89: Amino acid substitution matrices

Outline:

I. Interlude: Progress in solving the *Mystery of the Missing Match*
II. Scoring sequence alignments
III. Scoring tables for protein
IV. Practical *Blast*

I. Interlude: Progress in solving the *Mystery of the Missing Match*

I.A. Local *BlastN* vs NCBI's *BlastN*

Wednesday in class we discovered the following:

1. We have in our hands a working program *BlastN*, written in Perl, that at least in some respects acts like *BlastN* implemented by NCBI.
2. The parameters set in the local program are the same as those set in the NCBI implementation (except for *gap dropoff* and *Expect threshold*)
3. The main loop of the program sequentially extracts words from the query sequence and finds matches between them and the target sequence.
4. A subroutine called within the main loop is responsible for constructing the scoring tables necessary to extend the exact match in both directions.
5. The local implementation differs from the NCBI implementation in at least the following particulars:
    a. The local *BlastN* does not filter the query sequence
    b. The local *BlastN* does not calculate a score related to the probability of finding a match as good or better than the match found.
    c. Therefore, the local *BlastN* does not throw away or rank sequences based on this score.
6. Unlike NCBI-implemented *BlastN*, the local program printed several matches when DG47 (the sequence of the PCR product) was blasted against M29081 (bona fide *lef* gene). All the matches were very short, except for one that extended the length of DG47.

We have therefore yet another mystery: Why does local *BlastN* find the match between DG47 and *lef* that is apparent by eye while NCBI *BlastN* does not? It was suggested in class that the absence of filtering by local *BlastN* might be the answer.

SQ1. Test whether filtering is the key difference in two ways:

a. Compare DG47 and *lef* via NCBI's pairwise *Blast*, after disabling filtering (you can get both sequences from links in the notes for Sep 25).

b. Compare DG47 and *lef* via local *BlastN* with filtering. You could teach the program how to recognize and mask repetitive sequences, but for our purposes that's way too much work. Find the repetitive sequence yourself by eye and then edit the query to replace the repetitive sequence with random nucleotides. Use this modified DG47 also in a pairwise Blast search using NCBI's *Blast*.

I.B. Does local *BlastN* score properly?

Todd noticed that the way the Smith-Waterman algorithm handles gap penalties (which is the way we did it in class) may not be the way that Blast does it. The Smith-Waterman method charges a gap-opening penalty for the first gap and a gap extension penalty for each *additional* gaps. An alternative approach is to charge a gap-opening penalty *and* a gap extension penalty (or more accurately a gap-presence penalty) for the first gap and just a gap extension penalty for each additional gap. These two choices are illustrated in Table 1.

Table 1: Comparison of methods to calculate gap penalty

Method	**Formula for gap penalty**	**Example: penalty for** `AGGC` / `T--G` open → -5, ext → -2
Smith-Waterman	Gap_opening + gap_extension · (gap_size-1)	7
Alternative (Blast?)	Gap_opening + gap_extension · gap_size	9

Which method does NCBI *BlastN* use? We'll be able to answer this better after a discussion of scoring, but we can answer right now what method local *BlastN* uses.

SQ2. One way to tell is to examine the program and look for the code that calculates gap penalties. Find that part of the program. Where is it and which method does the program use?

SQ3. Another approach is to print out the scoring matrix and examine the values in adjacent boxes. The notes for Wed Sep 25 describes how to write a subroutine to print out some values of the scoring matrix. First of all, where in *BlastN* would you put a call to this subroutine (print_score) so that the values printed out are meaningful?

SQ4. Complete the subroutine print_score started for you in Wednesday's notes, insert it at the end of *BlastN*, and call it at the appropriate time (determined in SQ3). By the way, the notes for Wednesday have been modified so to contain a link for the starter program, incorporating some of the hints Fritz gave you.

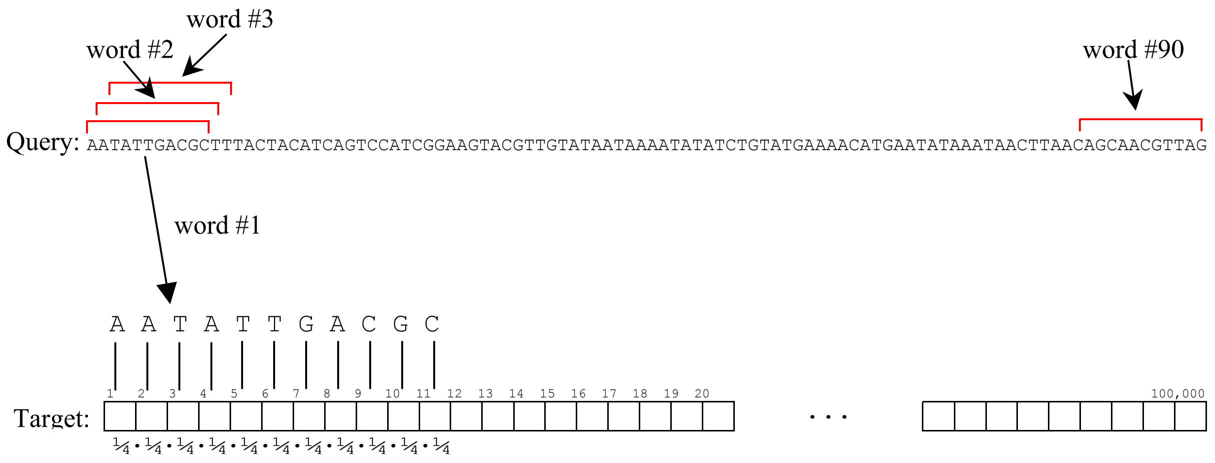
II. Scoring sequence alignments

Our version of *BlastN* prints out every word match it finds, after extending it in both directions as far as it can. Real *BlastN* doesn't do that. Instead, it calculates a score for each match, ranks all matches by their scores, and provides the top **x** matches that exceed a threshold **y** (you can specify **x** and **y**). The score thus serves to order the matches, so you can look at the best ones first, and also tosses out matches that are worse than you want to consider.

What kind of scoring system would be appropriate for sequence alignments? The first idea that might come to mind would be to use directly the raw score that *BlastN* calculates, i.e. one of the numbers found in scoring tables. This is OK for ranking matches, but suppose that *all* of the matches were no good, according to your tastes. How could you prevent *BlastN* from providing you with pages of garbage?

A better idea is to transform the raw score into a probability. If a match would occur frequently in a random sequence, you don't want to know about it. If you'd have to go through 10^{180} random sequences to find a match as good as the one *BlastN* is considering, then you want the program to pass that one on to you. So, we need to talk about expected frequency.

Suppose you're following Blast as it compares a 100-nucleotide sequence with a 100,000-nucleotide sequence. The first step is to extract a word from the query (let's say that the word size had been set to 11 nucleotides). What's the probability that this particular 11 nucleotides match the first 11 nucleotides of the 100,000-nucleotide sequence? Not much, sure, but we need a number. Let's simplify the situation and say that the probability of a match of one nucleotide is ¼ . If so, then:



The probability of a match between word #1 and positions 1-11 of the target is $(¼)^{11}$. But there is the same probability of a match between word #1 and positions 2–12 and 3–13, etc., all the way up to positions 99,990 – 100,000. That's about 100,000 possible matches, each with a probability of $(¼)^{11}$, giving a combined probability of $100{,}000 \cdot (¼)^{11}$. But that's not the end: there are also words #2, #3, all the way up to word #90,[a] so there are now $90 \cdot 100{,}000$ ways of getting a match, each with a probability of $(¼)^{11}$. That gives a total expected number of matches of:

$$90 \cdot 100{,}000 \cdot (¼)^{11}$$

[a] Why not 100 words? Because the last word <u>ends</u> at position 100, so must begin at position 90. The number of words = length – wordsize + 1.

or more generally:

(1) $\mathbf{E} = \mathbf{m} \cdot \mathbf{n} \cdot \mathbf{p}^{\mathbf{S}}$

where **E** is the expected number of matches, **m** is the effective length of the query, **n** is the effective length of the target, **p** is the probability of a single match, and **S** is (for the moment) the total number of matches.

Mathematically, it is more convenient to transform this expression into an exponential based on *e*, forcing the expression to look like:

(2) $\mathbf{E} = \mathbf{m} \cdot \mathbf{n} \cdot e^{-\lambda \mathbf{S}}$

To fit this form, it works out that the constant λ has to equal $-\ln(\mathbf{p})$.[b]

We now have almost what we want: a score that tells us how frequent we should expect a match to be. Unfortunately for this expression, *BlastN* finds matches that are more complicated than a string of matching nucleotides. This means that the expression has to be modified. I don't understand half the math necessary to tell you how the ultimate expression arises, but you can see that this expression used by *BlastN* to calculate expected frequency is pretty close to the expression we derived intuitively from first principles. In the actual expression:

(3) $\mathbf{E} = \mathbf{K} \cdot \mathbf{m} \cdot \mathbf{n} \cdot e^{-\lambda \mathbf{S}}$

where **K** and λ are empirically derived constants, the expected frequency, **E**, depends on the effective lengths of the query and the target, and it is related exponentially to the score **S**. Furthermore, λ is generally close to ln(¼), and **K** (a constant that depends on the base composition of the target) is not far from 1.

SQ5. Let's see these symbols in action. Go to the NCBI web site and do a *BlastN* search of M29081 against all available nucleotide sequences (see instructions in Scenario 3 if you've forgotten how to do this). Look at the very end of the results you obtain to find values for K, λ, m, and n. Then go to any match in the results (you'll have to scroll down from the top past the summary list) that has an E-value greater than zero. You'll find the E-value and the raw score, S, in the line that begins:

Score = nnn bits ([raw score]), Expect = [E-value]

Determine from these values whether the given E-value can be calculated by equation (3).

You'll notice in the same line that gave you raw score and **E**-value, there's a quantity called *bits*. This is yet another way of representing the score. One problem with the raw score is that its significance depends on other parameters (**K** and λ) that can change depending on the base composition of the sequence being examined. One way to transform the raw score into a score whose significance is independent of these quantities is to force the equation for **E**-value into the form:

(4) $\mathbf{E} = \mathbf{K} \cdot \mathbf{m} \cdot \mathbf{n} \cdot e^{-\lambda \mathbf{S}}$

(5) $\quad = \mathbf{m} \cdot \mathbf{n} \cdot 2^{-\mathbf{S}'}$

[b] $\mathbf{m} \cdot \mathbf{n} \cdot \mathbf{p}^{\mathbf{S}} = \mathbf{E} = \mathbf{m} \cdot \mathbf{n} \cdot e^{-\lambda \mathbf{S}}$, leads to $\mathbf{p}^{\mathbf{S}} = e^{-\lambda \mathbf{S}}$, and (taking the log of both sides) $\ln(\mathbf{p}) = -\lambda$.

leading to:

$$(6)\ \mathbf{S'} = [\,\lambda \cdot \mathbf{S} - \ln(\mathbf{K})\,]\ /\ \ln(2)$$

where **S'** represents the normalized score. Since the **E**-value depends in the latter expression only on **S**' and on readily determined quantities (i.e., the effective length of the query sequence, **m**, and the effective length of the target sequence,**n**), it is more transportable. Secondly, since **S**' is present as an exponent of 2, it relates the number of binary digits, or bits, necessary to represent a number, just as the exponent in 10^n tells you how many digits are in the decimal representation of a number. Bits is also the currency of information theory, making it a theoretically attractive unit, one that is introconvertible with the **E**-value. You might also think of bits as the number of binary digits it takes to represent the probability of a specific match as good as the one reported. Multiply this probability by the sizes of the two sequences considered and you get, as in equation (1), the expected number of matches.

SQ6. Using the same *BlastN* result from SQ1, calculate the E-value from the bit score (S') and equation (5), and determine whether this value corresponds well with the given E-value.

II. Aligning protein sequences

Aligning pairs of protein sequences introduces additional complications, although the search algorithm is only a bit different from that used in aligning nucleic acid sequences. The root of the problem is the more complex functional roles played by amino acids relative to nucleotides. With nucleotides, scoring is simple: there's a gap, a match, or a mismatch. With amino acids, there are various levels of mismatch, as illustrated in Figure 1. The region of the protein shown in the figure is relatively well conserved over evolution, but there is variation in amino acids at each position, and there is often order to that variation. For example, the first position is occupied by either lysine or arginine, both positively charged amino acids. The second position shows either isoleucine, leucine, or valine, all aliphatic[c] hydrophobic amino acids. The last position is a different matter, however, with a mixed bag of amino acids. One would imagine that in many cases, mutation from one amino acid to a similar amino acid would retain function of the protein, and we would like to score that mismatch differently from a mutation that leads to a more drastic change.

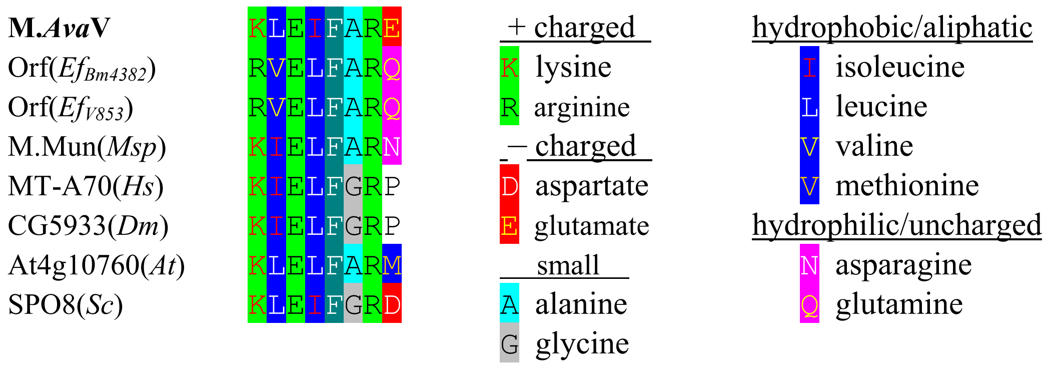


Figure 1: Alignment of a region from eight DNA-modifying enzymes (some putative). Note that the alignment isn't perfect, but most of the imperfections fall into certain families of amino acids.

[c] An "aliphatic" amino acid is one means that has a side chain consisting solely of a hydrocarbon chain.

II.A. Percent Accepted Mutation (PAM) tables

We would like to be able to look at a protein and say that this amino acid change is minimally significant, hence a low mismatch penalty, while another is worthy of a high mismatch penalty, but we don't yet know enough about protein structure to do this. Instead, we've asked the proteins to tell us to score their own mutations. This has been done in two ways. Margaret Dayhoff collected pairs of very similar protein sequences that differed from each other in about one amino acid per one hundred resides and analyzed which amino acid were replaced by which others. All of the proteins analyzed were functional, so the amino acid changes did not abolish function and were termed "acceptable". The resulting table, called PAM1 for *P*ercent *A*ccepted *M*utation at *1*% change, provides a guide as to what amino acid substitutions might be considered conservative (more likely to retain function of the protein) and a basis for the construction of a rational scoring table for protein sequence alignment.

However, a table constructed from the comparison of closely related protein is not as applicable to the comparison of proteins that have a much greater degree of divergence. For this reason other PAM tables were calculated. Multiplying the PAM1 matrix by itself gives a PAM2 matrix, representing predicted mutation frequencies amongst proteins with a 2% degree change. By this procedure, matrices as high as PAM400 have been calculated. It may seem surprising that there exist a table that predicts the frequencies of amino acid substitutions in protein with 400% degree of change, but you have to remember that over millions of years, the same position in a protein may change multiple times. A 400% degree of change means that each amino acid position has changed, on average, 4 times. Figure 2 provides a theoretical correlation between the degree of change and the percent similarity between protein. PAM tables higher than PAM250 are of limited value, because with such a low degree of similarity, matches between amino acids may have arisen by chance rather than common descent. PAM250 is commonly used in Blast searches.

Table 2: PAM1 amino acid change frequencies*

		Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr
		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y
Ala	A	9867	2	9	10	3	8	17	21	2	6	4	2	6	2	22	35	32	0	2
Arg	R	1	9913	1	0	1	10	0	0	10	3	1	19	4	1	4	6	1	8	0
Asn	N	4	1	9822	36	0	4	6	6	21	3	1	13	0	1	2	20	9	1	4
Asp	D	6	0	42	9859	0	6	53	6	4	1	0	3	0	0	1	5	3	0	0
Cys	C	1	1	0	0	9973	0	0	0	1	1	0	0	0	0	1	5	1	0	3
Gln	Q	3	9	4	5	0	9876	27	1	23	1	3	6	4	0	6	2	2	0	0
Glu	E	10	0	7	56	0	35	9865	4	2	3	1	4	1	0	3	4	2	0	1
Gly	G	21	1	12	11	1	3	7	9935	1	0	1	2	1	1	3	21	3	0	0
His	H	1	8	18	3	1	20	1	0	9912	0	1	1	0	2	3	1	1	1	4
Ile	I	2	2	3	1	2	1	2	0	0	9872	9	2	12	7	0	1	7	0	1
Leu	L	3	1	3	0	0	6	1	1	4	22	9947	2	45	13	3	1	3	4	2
Lys	K	2	37	25	6	0	12	7	2	2	4	1	9926	20	0	3	8	11	0	1
Met	M	1	1	0	0	0	2	0	0	0	5	8	4	9874	1	0	1	2	0	0
Phe	F	1	1	1	0	0	0	0	1	2	8	6	0	4	9946	0	2	1	3	28
Pro	P	13	5	2	1	1	8	3	2	5	1	2	2	1	1	9926	12	4	0	0
Ser	S	28	11	34	7	11	4	6	16	2	2	1	7	4	3	17	9840	38	5	2
Thr	T	22	2	13	4	1	3	2	2	1	11	2	8	6	1	5	32	9871	0	2
Trp	W	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	9976	1
Tyr	Y	1	0	3	0	3	0	1	0	4	1	1	0	0	21	0	1	1	2	9945
Val	V	13	2	1	1	3	2	2	3	3	57	11	1	17	1	3	2	10	0	2

*Each cell gives the number of times the amino acid listed horizontally changed to the amino acid listed vertically, per 10,000 changes of the first amino acid.

SQ7. Amongst protein pairs that are 99% similar to each other, what fraction of arginines in one protein correspond to lysines in the other (at the equivalent position)? What fraction of arginines in one correspond to leucines in the other?

SQ8. What PAM table would be appropriate to search for proteins about 50% identical to a query sequence?

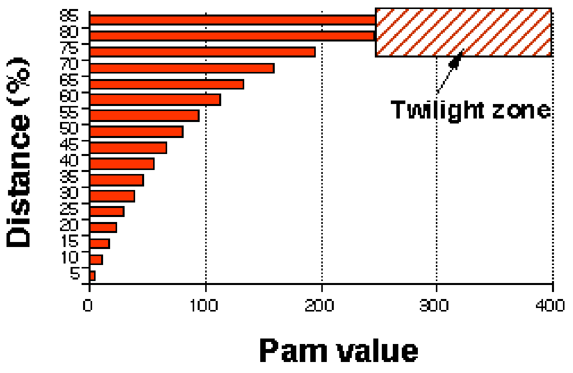


Figure 2: Relationship between degree of change (Pam value) and dissimilarity between two proteins. The hatched box indicates the degree of dissimilarity sufficient to cast doubt on a conclusion of common lineage between two proteins. (*from Doolittle, 1987*)

II.B. Block Substitution Matrices (BLOSUM)

PAM tables suffer from several deficiencies. Since they're all calculated from PAM1, any error in that table is propagated throughout the rest. Since the original data consisted of only 1572 amino acid changes, many possible changes were not observed (e.g. aspartate to arginine). Even though such changes must be possible at some frequency, no PAM table will not admit this. Secondly, the procedure of multiplying PAM matrices to get tables for less similar proteins presumes the debatable evolutionary model in which similar and dissimilar protein pairs arose from the same kind of amino acid substitutions. Third, PAM tables are built from a consideration of amino acid substitutions at all positions in similar proteins. It may well be that the pattern of amino acid substitutions would differ in different regions.

Henikoff and Henikoff (1992)[d] proposed an alternative strategy to address the deficiencies in PAM tables. They compiled a list of conserved blocks of amino acid sequences (regions within proteins that are relatively conserved across many organisms) and calculated the frequency of each amino acid change. Owing to the size of their database, even the least common amino acid transitions is represented 2369 times. The table derived from blocks in which there is 62% identity of amino acids is called BLOSUM62 (Block substitution matrix, 62%). Two other tables, BLOSUM45 and BLOSUM80 are also available, derived from blocks with 45% and 80% identity, respectively. Note that BLOSUM's numbering system is the opposite in direction from PAM's. Per BLOSUM: low numbers mean low identity. Per PAM: low numbers mean low *change* (so *high* identity).

II.C. Constructing scoring matrices from PAM and BLOSUM tables

Blast uses PAM and BLOSUM tables to calculate a probability that a series of matches and mismatches would occur given the presumed model. To facilitate this calculation, the tables are modified in the following way.

1. The frequency of each amino acid-amino acid transition is rendered as an odds ratio. For example, for the transition from asp-glu (in either direction):

$$\frac{\text{Total number of asp-glu transitions observed/Total number of transitions observed}}{2 \cdot (\text{frequency of asp residues}) \cdot (\text{frequency of glu residues})} = \frac{\text{relative frequency}}{\text{expected frequency}}$$

[d] Henikoff S & Henikoff J (1992). Proc Natl Acad Sci USA 89:10915-10919.

The expected frequency is the fraction of transitions you'd expect just from the frequency of the component amino acids. So the odds ratio is the degree to which the frequency of an amino acid transition deviates from expectation.

2. The from the odds ratio, the log odds ratio (**lod**) is derived:

   $$\textbf{lod} = \log_{10}(\text{relative frequency/expected frequency})$$

   Adding the **lod** scores is equivalent to multiplying the odds ratios, and since addition is faster than multiplication, the program becomes more efficient.

3. The **lod** score is multiplied by 10 and only the integer portion is shown. This makes the table easier to read, though making it more difficult to comprehend how the number relates to the odds ratio.

Figure 3 shows the BLOSUM62 table in its log odds ratio form. A scoring matrix would use this table to replace match rewards and mismatch penalties. The alignment of the first sequences given in Figure 1:

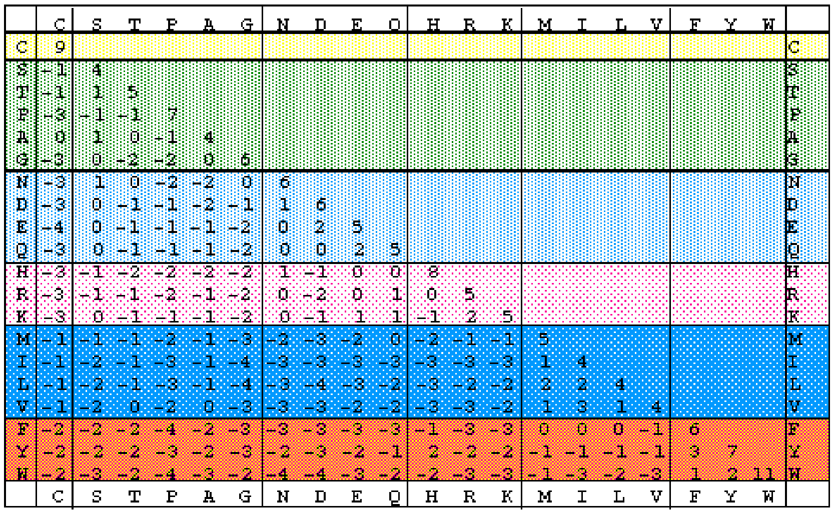
KLEIFARE

RVELFARQ

would give a score (using BLOSUM62) of:

2+1+5+2+6+4+5+2 = 27

Gaps are still dealt with using gap open and gap extension penalties as in *BlastN*.



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

Figure 3: BLOSUM62. Each number represents the $\log_{10}$ of the odds that a given amino acid transition occurs. Since the table is symmetrical (the direction of change is irrelevant), only half of it is shown.

SQ9. What is score do you calculate for the alignment between the last two sequences given in Figure 1?

II.D. One additional difference between *BlastP* and *BlastN*: The initial match

BlastN begins by looking for an exact match in the target sequence of a word extracted from the query sequence. Unfortunately, exact matches in proteins are too rare for this procedure to be useful. Instead, *BlastP* (analogous to *BlastN*, but acting on protein sequences) searches for exact and *almost* exact matches to words. How exact the match has to be is determined by what is known as the neighborhood word threshold value (**T**). For example, if **T** is set to 13, then the word **FIM** will find the following sequences: **FIM** (score=15), **FIM** (score=13), and **FLM** (score=14). Any of these three sequences in the target will instigate a search for extensions in both directions.

SQ10. What sequences would be found by VLI using a T value of 13? Considering your answer, do you think that this system or the T value used has a defect?

III. Practical *Blast*

You now know how to score DNA and protein alignments, you have an idea how *BlastN* and *BlastP* work… how does this change your life? If you do alignments (and there's probably nothing more common in bioinformatics than that), you should now be able to talk to *Blast* better, to make it do what you want.

III.A. Which scoring matrix to use? (protein searches only)

From the considerations discussed above, you shouldn't be surprised to learn that BLOSUM matrices outperform their PAM counterparts, so if you are searching for protein that figure to be around 45%, 62%, or 80% identical, then choose BLOSUM45, BLOSUM62, or BLOSUM80, respectively (BLOSUM62 is *BlastP*'s default). If you're using a short query, then you want to choose a matrix that demands a higher degree of identity (otherwise true matches will get mixed in with the background). In that case, PAM30 (length <35 amino acids) or PAM70 (length between 35 and 50 amino acids) is preferable to any available BLOSUM table.

III.B. What scoring parameters to use?

If you want to find the most significant matches, leave the Blast default parameters alone. You'll often get a patchwork of partial matches between your query and the target, as shown in Figure 4. Often it is possible to connect the strong matches with less significant matches by reducing the gap penalties or reducing the mismatch penalty (in the case of nucleotides).

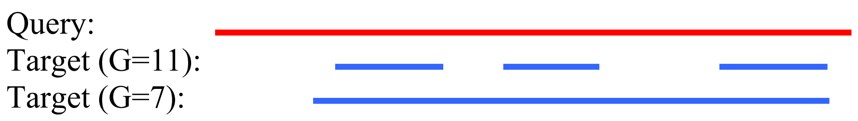


Figure 4: Effect of changing gap open penalty (G). Hypothetical example of results of a Blast search of a query sequence (red line) against a target sequence. Regions found by the search are shown in blue.

If you're searching for short matches, you may want to reduce the word size and increase the E-value threshold (since even good short matches will get worse scores than the long matches Blast is set up for). If you're looking for significant matches and you're getting a lot of them, you can reduce their number by setting the E-value from 10 (default) to .001. While this will almost always get rid of many spurious matches, you will need to decide if anything valuable is also being tossed out.

Probably the common mistake people make with Blast is to search for DNA similarity when they could instead look for protein similarity. Evolution operates at the level of protein function, not DNA sequence, so DNA searches invariably miss distant matches that protein searches find.

The second common mistake people make with Blast is to equate similarity of sequence with similarity of function. This is a large topic that takes us out of the scope of this course, but you might heed the warning given by Persemlidis and Fondon.[e]

[e] http://genomebiology.com/2001/2/10/reviews/2002