

CS 124 Spring 2005 Midterm

The exam is open book and notes, but only from materials we used in this class. Write your answers on the midterm. Use the back if needed. Write clearly and to the point. Don't copy verbatim from the notes. The numbers in parenthesis are the estimated maximum number of minutes needed for the problem and the point score for the problem.

1. (10,10) Show the DP table used to find an optimal local alignment between the strings: $S_1 = \text{ATTGAC}$ and $S_2 = \text{TTAGAG}$. Use +1 for a match, and -1 for a mismatch or a space. There is no term for a gap in this objective function. Put S_1 on the side (vertical) axis and S_2 on the top (horizontal) axis and write big. Use the back of the page. Be sure to show all the proper backpointers, and then use them to find and display an actual optimal local alignment.

Solution: Before drawing the table, we will briefly state the definitions of *global* and *local* alignment (because a common mistake was to calculate the wrong alignment).

- *Global alignment* Given two strings S_1 and S_2 , the *optimal global alignment value* or *similarity* is the value obtained by allowing spaces inside or at the ends of both strings (with no two spaces opposite each other), and then calculating the alignment value.
- *Local alignment* Given two strings S_1 and S_2 , find substrings α and β of S_1 and S_2 respectively, such that the similarity (optimal global alignment value) of α and β is maximal over all pairs of substrings.

Thus, for local alignment, we have the following base case:

$$\begin{aligned} V[i, 0] &= 0 \\ V[0, j] &= 0 \end{aligned}$$

Remembering that:

$$c[i, j] = \begin{cases} 1 & \text{if } S_1[i] = S_2[j] \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

we have the following recurrence relation:

$$V[i, j] = \text{MAX} \begin{cases} 0 \\ V[i-1, j-1] + c(i, j) \\ V[i-1, j] - 1 \\ V[i, j-1] - 1 \end{cases} \quad (2)$$

Our table is calculated as follows:

	0	T	T	A	G	A	G
0	0	0	0	0	0	0	0
A	0	0	0	1	0	1	0
T	0	1	1	0	0	0	0
T	0	1	2	1	0	0	0
G	0	0	1	1	2	1	1
A	0	0	0	2	1	3	2
C	0	0	0	1	1	2	2

In order to find the optimal local alignment, we must find the largest value in *any* cell in the table, and then trace back from that cell until we encounter a cell with value 0. Thus, according to this table, the largest cell is $V[5, 5] = 3$ and we trace back until $V[1, 0] = 0$, to find the following maximal local alignment:

T	T	A	G	A
T	T	-	G	A

2. (10,10) Suppose we have sequences of analogous genes (genes that have essentially the same function) in a variety of organisms - for example suppose we have the gene for myoglobin in humans, chickens, horse, seahorse, housefly etc. Those gene sequences are not identical. By comparing those sequences two at a time, we want to find which pairs of the gene are highly “biologically related”.

Some people have suggested using a) the length of the longest common substring between two strings to “measure” or “expose” the level of “biological relatedness” of two strings. Others have suggested using b) the length of the longest common subsequence; others have suggested using c) global alignment with a user defined substitution matrix for the values and costs of specific matches and mismatches, and no term for gaps; and others have suggested using d) the value of the optimal global alignment between the

strings, with a value of 1 for a match, and a cost -1 for either a mismatch or a space, and no term for gaps in the objective function.

Order the above four suggested approaches (a,b,c and d) in the terms of their effectiveness in measuring or exposing the “biological relatedness” of two gene sequences. Be clear to indicate the direction of the order, i.e., which end represents the most effective approach and which is the least effective.

Explain why you chose that ordering.

Answer: Actually, there were several possible orderings that could be justified, and the grading was based on how well you supported the order that you proposed. The two relations that are most easily supported are that a) (Longest Common Substring) is better than b) (Longest Common Subsequence); and that c) global alignment with a substitution matrix was better than d) global alignment with constant values for every matches and mismatches. Certainly c) is better than d) because d) is a special case of c) i.e., one can always create a substitution matrix that enforces the constant values used in d), but c) also allows more sensitive and selective alignments through the use of the substitution matrix. a) is (almost always) better than b) because the longest common subsequence between random strings is relatively long (more than $n/2$ for DNA), so there is little room for values that distinguish non-random relationships from random ones. On the other hand, the expected length of the Longest Common Substring for random strings is only about $\log n$, which leaves a lot of room for larger scores. Moreover, there are many biological phenomena (recall all the motivation for local alignment) that lead to the expectation that there will be a fairly long common substring between related sequences.

So the relationship of a) better than b) and c) better than d) were expected and the easiest to justify. Many people also said that c) and d) are better than a) and b) and that is the relationship I think is easiest to justify. The key is that the problem stated that we are looking at analogous genes, so there should be a fair amount of sequence similarity throughout the full extent of the sequences. That is, if one concatenates the exons in each gene, then the sequences will have high similarity on both the right and left ends, and in the middle. The intron may not be highly similar, but given the four choices stated in the problem, global alignment should be able to highlight the most related sequences. However, if you answered that because of introns, you felt that global alignment would not be effective, and so you thought that a) was best, and you made a good case for it, that was accepted also.

One mistake that many people made was to say that c) and d) did not allow gaps in the alignment, and so they would not be effective alignments. c) and d) do allow gaps, but there is no explicit term for gaps in the objective function.

Problem 3

(10, 15) One type of alignment that we did not talk about in class is where string S_1 must be aligned to “inside” S_2 . Another way to say this is that either end of S_2 can be in an end-gap in the alignment, with no cost, but neither end of S_1 can be in an end-gap in the alignment. Suppose each match has value 1 and each mismatch or counted space has cost -1. For example, if $S_2 = ATTCGGA$ and $S_1 = TGGAA$, then the alignment

```
ATTCGGA
  TG-GAA
```

puts S_1 inside S_2 and has cost zero.

Explain how to modify the global alignment DP recurrences to find the best alignment where S_1 must be aligned inside S_2 . Be sure to specify the base cases, the recurrences, where the optimal value is found in the DP table and how to find the actual alignment (not just its value).

Solution: In order to insure that S_1 is aligned inside S_2 , we must establish a base case and recurrence relation that forces a huge penalty onto S_1 if it tries to align “outside” of S_2 . Thus, our base case is as follows:

$$\begin{aligned} V[i, 0] &= -\infty \\ V[0, j] &= 0 \end{aligned}$$

This base cases forces a penalty on S_1 for initially aligning with a space, but it does not penalize for extending beyond the end of S_2 . We must take care of that case within the recurrence relation. Remembering that:

$$c[i, j] = \begin{cases} 1 & \text{if } S_1[i] = S_2[j] \\ -1 & \text{otherwise} \end{cases} \quad (3)$$

we have the regular global alignment recurrence relation, with one modification:

$$V[i, j] = \text{MAX} \begin{cases} V[i-1, j-1] + c(i, j) \\ V[i-1, j] - 1 & \text{if } j \neq n \\ -\infty & \text{if } j = n \\ V[i, j-1] - 1 \end{cases} \quad (4)$$

Thus, we penalize for moving vertically in the last column, which prevents S_1 from extending past the end of S_2 . The value for the optimal alignment is found in $V[n, m]$, and the alignment itself is found by tracing back from that cell. Thus, for the example given above, with $S_1 = TGGAA$ and $S_2 = ATTCTGGA$, we have the following table:

	0	A	T	T	C	G	G	A
0	0	0	0	0	0	0	0	0
T	$-\infty$	-1	1	1	0	-1	-1	-1
G	$-\infty$	-2	0	0	0	1	0	-2
G	$-\infty$	-3	-1	-1	-1	1	2	-1
A	$-\infty$	-4	-2	-2	-2	0	1	1
A	$-\infty$	-5	-3	-3	-3	-1	0	2

Thus, according to this table, we see that the optimal alignment (of cost 2) is as follows:

A	T	T	C	G	G	-	A
-	-	T	-	G	G	A	A

4. (10,10) In the Granin story in the class notes (which you were instructed to read in preparation for this midterm), the people who claimed there was an informative match between the BRCA1 sequence and the Granin motif (basically a regular expression for a set of sequences), made a fundamental mistake in dealing with probabilities. We have discussed that kind of mistake in class several times. What was the general mistake, and how did we specifically illustrate it in class?

Solution: The general mistake is to calculate the probability of the specific result obtained from the database search, rather than calculating the probability of obtaining a result as good or better than the one obtained. In the granin case they calculated the probability that a random string (with the composition of BRCA1) would match the regular expression for Granins.

That probability is very low. But they should have calculated the probability that a random string (with the composition of BRACA1) would have matched some regular expression in the Prosite database. That probability is very high.

We discussed such phenomena in class in several ways. For example, in database searching generally, suppose you find a string in the database that aligns with the query string, with a score of S . In order to evaluate the significance of that alignment, you cannot just compute the probability that a random string would align with the query string with score S (a probability that is likely to be low), but rather you have to take into consideration that the database has a huge number of strings. This is the “multiple testing” problem. So you have to compute (or assess somehow) the probability that the query string would align to some string in a random database (with the composition of the real database) with score of S or better. That probability may be very high. To make that assessment, you can randomize the real database and then do alignments of the query string to that randomized database, or use the Bonferoni (or Phony-Balonne) correction, or use analytical results if they are known.

Another way we discussed this was to consider flipping a coin 100 times. The probability of the result (considered as the actual sequence of heads and tails obtained) is $\frac{1}{2^{100}}$, which is tiny. But does that mean that the sequence can not be explained as a random event? No, because, every specific sequence has that probability of occurring, and so the probability of obtaining a result with that “score” is exactly one.

5. (10,10) Explain in your own words why sometimes we need to give each gap in an alignment an explicit cost. Why not just give costs for each space? Is this more important in aligning DNA or Protein? Explain.

Answer: In this question again, there were several ways to answer the question, but a full answer should have discussed both biology, and the way alignments are influenced by their objective function. For the biology side, there are many phenomena (introns, protein domains, exterior loops, transposons,) that create contiguous sequences that are either inserted into or deleted from a sequence. Therefore when aligning the sequences with and without those intervals, one will see a long gap. We want our alignment methods to be able to create that same gap, and without an explicit term for gap in the objective function, there is no difference between k spaces that are spread out along an alignment, and k spaces that are organized into a con-

tiguous interval. We need an explicit term in the objective function for gaps in order to encourage the alignment to choose a few gaps instead of many dispersed spaces. Of course, how effective this will be depends on the gap cost v. space cost etc., but with an explicit term for gaps, we have a chance of creating the right parameters to model the given biological phenomena that involves long inserts and deletes.

6.(5,5) BLAST is a tool that tries to find a few sequences in a database that are “sufficiently biologically related” to the query sequence. In that context, what does it mean to say that BLAST produces a “false positive” or a “false negative” result? In general, if you want to reduce the number of false negative results, should you set Expect parameter (the E value threshold for reporting results) in BLAST to be large or small? Explain.

Solution: A *false positive* is when BLAST reports a sequence that has a high similarity value with the query string, but no real biological relation to the query string. A *false negative* is when BLAST does **not** report a sequence that has a real biological relation to the query string. In order to reduce false negatives, one should **raise** the E value (remember that the E value for an alignment score S represents the number of hits with a score $\geq S$ that would be “expected” by chance when searching a database of a particular size). Thus, a larger E value means more hits, which means more strings, which means less false negatives.

7. (5,10) Explain what the following Perl program does. I am not interested in what each statement does, but what the entire program does. That can be stated in one or two lines.

```
#Mystery Program
#
$string1 = <>;
$string2 = <>;

@str1 = split(//,$string1);
@str2 = split (//,$string2);

$len1 = @str1;
$len2 = @str2;
```

```

$i1 = 0;
$i2 = 0;
while (($i1 <= $len1) and ($i2 <= $len2)) {
    if ($str1[$i1] eq $str2[$i2]) {
        $i1++;
        $i2++;
    }
    else {
        $i2++;
    }
}

    if ($i1 > $len1) {
        print "Yes it is\n";
    }
    else {
        print "No it is not\n";
    }
}

```

Solution: This program walks S_1 and S_2 , incrementing pointers in both-strings whenever a matching character is found, otherwise incrementing only the pointer for S_2 . Thus, the program returns *yes* if S_1 is a subsequence of S_2 and returns no otherwise.

8. (10,10) Write a Perl program that reads in lines of text from a file named “textfile” and then checks each line to see if the line starts with a text label consisting of 2 or 3 alphabetic characters, then the line has a space, and then just 0’s and 1’s for the rest of the line. When the program finds such a line, it removes the text label and the space in that line, and prints out rest of the line. It does not print out other kinds of lines. For example, if the textfile has the following six lines:

```

Start of File
IDa 0010010011
The next SNP is
IDb 1101010110
IDc 0000100100
End of File

```

then the output would be:

```
0010010011
1101010110
0000100100
```

Solution: While there are many possibilities for this program, here is one that works:

```
#!/usr/bin/perl -w
#
#
open IN, "textfile";
while ($line = <IN>) {
    if ($line =~ m/^[A-Za-z]{2,3}\s([01]+)/) {
        print "$2\n";
    }
}
}
```