Midterm Exam Solutions

CS 181, Fall 2014

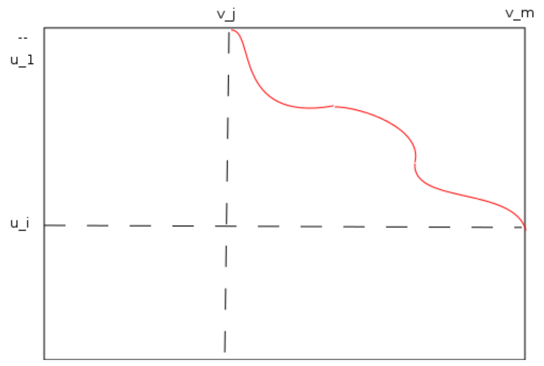
Problem 1: Sequence Alignment

1. Computing the edit distance between two strings. **Global alignment**
2. Aligning 30 similar sequences of length 2kb in a reasonable amount of time. **Star alignment** (not DP multiple alignment, which requires exponential memory)
3. Aligning short reads ($\sim$100 bases) against a reference genome ($\sim$3 billion bases). **Fitting alignment**
4. Designing a 30 base-pair single-stranded DNA segment and you want to minimize binding to itself. **Nussinov algorithm**
5. Computing the longest common substring of two strings. **Local alignment** (not semiglobal alignment)
6. Aligning two long sequences on a computer with limited memory. **Global, space-efficient alignment**
7. Efficiently aligning the genome of a virus to that of a closely related virus. **Global, banded alignment**
8. Quickly finding an approximately optimal alignment of one string to another. **Global alignment with four-russians**
9. Efficiently creating an optimal multiple alignment of three strings (NOT necessarily similar strings). **DP multiple alignment with Carrillo-Lipman** (not star alignment, which is a heuristic that doesn't guarantee an optimal alignment, whereas Carillo-Lipman does).

Problem 2: Prefix-Suffix Alignment

It is helpful to think backwards to the fitting alignment problem. In fitting alignment, we had to find an optimal global alignment over all substrings of u to *the entire string* v. In the dynamic programming table, we had to design our algorithm so that we could *enter* at any row we wanted, but we were forced to exit *somewhere* along the last column, since we had to use *all* of v.

Using that same idea, we could design a dynamic programming table where we can *enter* at any column, but only on the first row. On the other hand, we **must** exit on the last column, but we may do it on any arbitrary row. Shown on the right is a figure illustrating this idea.



(a) In the dynamic programming table S, each cell $S(i, j)$ will represent the optimal prefix-suffix alignment score of the ith prefix of u and the jth prefix of v. (This indicates that we will be using S to represent the characters of u using the rows, and v using the columns.)

An alternative, but equivalent way to define it, is to say that $S(i, j)$ = the highest global alignment score between the ith prefix of u, and *all* substrings of v that ends in j.

Note: A LOT of people said $S(i, j)$ is the best alignment score of the ith prefix of u and the jth suffix of v, which is incorrect. One way to realize why this is not the right thing to do, is to notice that if we fill in the table starting from the top left, there is no way to have information about the jth suffix of v, since we haven't looked at it yet.

(b) Given our setup in (a), we initialize S by setting every cell in the first row to 0, and every cell in the first column, starting from $i = 1$, using the simplified recurrence $S(i, 0) = S(i - 1, 0) + \delta(v_i, -)$. *Note:* Alternatively, it is normally safe to assume that a gap penalty is a fixed constant σ, so we could also set the first column entries $S(i, 0)$ to $\sigma \cdot i$.

(c) The recurrence relation is, surprisingly enough, exactly the same as the global alignment problem's recurrence relation.

$$S(i, j) = \begin{cases} S(i-1, j) + \delta(u_i, -) \\ S(i, j-1) + \delta(-, v_j) \\ S(i-1, j-1) + \delta(u_i, v_j) \end{cases}$$

(d) To find the best prefix-suffix alignment score, we look for the highest score on the last column of the table. In other words, we look for the i that maximizes $S(i, m)$, where m is the length of the second string, v.

Problem 3: Sankoff's Algorithm

First, consider trait 1. The table for node 2 is

+	-	o
1	1	5

The table for node 1 is

+	-	o
3	4	3

+ would point to + for node 2, - would point to -, and o would point to +.

Thus, the possible assignments are node 1: +, node 2: +; or node 1: o, node 2: +. The score is 3.

Now, consider trait 2. The table for node 2 is

b	m	s
3	1	1

The table for node 1 is

b	m	s
2	2	3

With b pointing to m, m pointing to m, and s pointing to s.

Thus, the best assignments are m or b for node 1 and m for node 2. The score is 2.

The total score for both traits would be 5.

Problem 4: Phylogeny Construction

(a) You are given the following distance matrix, D, and want to reconstruct the corresponding phylogenetic tree and its branch lengths.

	A	B	C	D
A	0	11	10	9
B	11	0	3	12
C	10	3	0	11
D	9	12	11	0

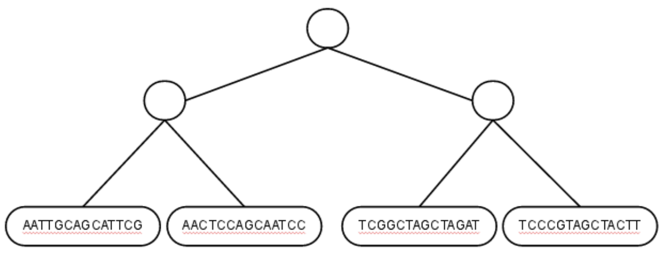
Using the 4-point condition, D is *additive*. This means that we can use both the **Additive Phylogeny AND Neighbor Joining** algorithms to reconstruct the phylogenetic tree and its branch lengths.

(b) You are given the following distance matrix, M, and want to reconstruct the corresponding phylogenetic tree and its branch lengths.

	A	B	C	D
A	0	6	8	18
B	6	0	2	13
C	8	2	0	10
D	18	13	10	0

Using the 4-point condition, M is *not additive*. This means that we can use **Neighbor Joining** but NOT Additive Phylogeny to reconstruct the phylogenetic tree and its branch lengths.

(c) You are given the following tree and want to fill in the missing node labels.



We can use **Sankoff's or Fitch's Algorithm** to label the internal nodes of the tree.

(d) You observe the following about the traits possessed by 5 species (A, B, C, D, E) and want to reconstruct the complete phylogenetic tree (including branch lengths and ancestral nodes).

Species A has wings, red eyes, four legs, and a brown abdomen.
Species B has wings, white eyes, two legs, and a brown abdomen.
Species C has red eyes, four legs, and a tan abdomen; species C is wingless.
Species D has wings, red eyes, four legs, and a tan abdomen.
Species E has white eyes, two legs, and a tan abdomen; species E is wingless.

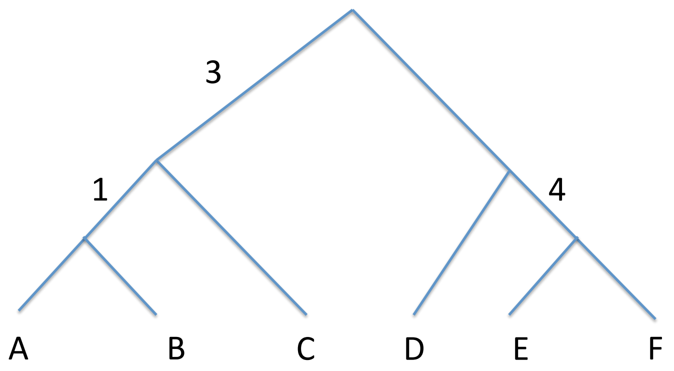
We can use **Perfect Phylogeny** construct a complete phylogenetic tree.
(Another way of approaching this problem would have been Neighbor Joining; but for full credit with that answer, you needed to have explained the meaning of "distance" between species and the method with which you would label internal nodes of the tree.)

Problem 5: Tree Splits

(a) $\{A\}|\{B,C,D,E,F\}$, $\{B\}|\{A,C,D,E,F\}$, $\{C\}|\{A,B,D,E,F\}$, $\{D\}|\{A,B,C,E,F\}$, $\{E\}|\{A,B,C,D,F\}$, $\{F\}|\{A,B,C,D,E\}$, $\{A,B\}|\{C,D,E,F\}$, $\{A,B,C\}|\{D,E,F\}$, $\{E,F\}|\{A,B,C,D\}$.

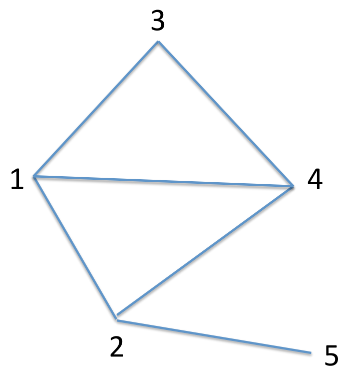
(b) **Note**: This question did *not* ask which characters were pairwise compatible, but rather which characters were compatible with the given tree T.

Characters 1, 3, and 4 are compatible with T, with edges labeled to show where character i changes state from 0 to 1.



Character 2 is not compatible with T because $2_1 = \{C,D\}$ is not a split in T. Character 5 is not compatible with T because $5_1 = \{B,F\}$ is not a split in T.

(c) Using the four gametes conditions, we compute the following compatibility graph.



(Sorry, I couldn't compile the bonus questions in time. Ask Prof. Raphael if you have questions about the bonus problems.)