



Problem Set 1: Sequence Alignment and Dynamic Programming

Due at the beginning of class on Wednesday, January 28.

Collaboration is allowed in groups of at most three students, but you must submit separate writeups. Please write the names of all your collaborators on your submissions.

If you are working alone, we will drop the problem with the lowest score and multiply your total problem set score by 4/3.

Problem 1. (25 points)

A *traceback path* is the backtracking path that follows the pointers to reconstruct the alignment in the dynamic programming matrix. Since the traceback paths in a dynamic programming table correspond one-to-one with optimal alignments, the number of distinct optimal alignments can be obtained by computing the number of distinct traceback paths.

- (a) (5 points) Prove that this number can be exponential in $|x|$ and $|y|$ by constructing sequences x and y that have exponentially many optimal (global) alignments, for fixed scoring parameters m , $-s$, and $-d$.
- (b) (7 points) Give an algorithm to compute the number of optimal alignments in $O(|x||y|)$ time.
- (c) (7 points) Derive the number of different alignments between two n -letter sequences. The answer may be left as a summation, but a closed-form solution also exists.
- (d) (6 points) There are 4^n different RNA molecules of length n . How many different DNA molecules of length n are there? Recall that a DNA molecule is double stranded, and is equivalent to its reverse complement.

Problem 2. (25 points)

Recall the linear-space global alignment algorithm which reconstructs the optimal alignment. In this problem, you will be asked to consider possible extensions of it to other alignment problems. For each part below, either briefly describe an extension of the linear-space algorithm to the version of the problem while preserving the space-time complexity, or discuss why this is not possible.

- (a) (7 points) Local alignment (Smith-Waterman), where you are asked to find a single optimal highest scoring local alignment.
- (b) (6 points) Alignment under affine gap scores. (Be careful with your answer)
- (c) (6 points) Alignment under general convex gap scores $\gamma(k)$.
- (d) (6 points) In the linear space alignment, the original problem of size $M \times N$ is reduced to two subproblems of sizes $kM/2$ and $(N-k)M/2$. In a parallel implementation of sequence alignment, it is desirable to have a *balanced partitioning* that breaks the original problem into two sub-problems of equal sizes. Design a linear space alignment algorithm with balanced partitioning.

Problem 3. (25 points)

A string $a = a_1 \dots a_j$ is a substring of a string $y = y_1 \dots y_n$, if for some $0 \leq i \leq n - j$, $y_{i+1} \dots y_{i+j} = a_1 \dots a_j$.

A string a is a subsequence of y if for some $1 \leq i_1 < i_2 < \dots < i_j \leq n$, $y_{i_1} y_{i_2} \dots y_{i_j} = a_1 \dots a_j$.

A string a is a supersequence of a string y if y is a subsequence of a .

A string a is a superstring of a string y if y is a substring of a .

(a) (6 points) The scoring function of an alignment problem plays an important role in determining the resulting alignment. Given two sequences x and y , describe explicitly what the Smith-Waterman algorithm finds for the following scoring schemes:

(i) $m = 1, s = 0, d = 0$.

(ii) $m = 1, s = \text{Infinity}, d = \text{Infinity}$.

(b) (6 points) Given strings x and y , devise an algorithm to find the *shortest supersequence* for both x and y .

(c) (6 points) Recall the recurrences for convex gaps: Given a convex gap penalty function $\gamma(n)$,

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(k, j) - \gamma(i-k) \text{ for } k = 0 \dots i-1 \\ F(i, k) - \gamma(j-k) \text{ for } k = 0 \dots j-1 \end{cases}$$

What is the problem with these recurrences if γ is not convex? As an example, consider

$$\gamma(1) = \gamma(2) = 1,$$

$$\gamma(k) = 1000 \text{ for } k \geq 3$$

(d) (7 points) Give correct recurrences for a general (not necessarily convex) γ , and make sure that your recurrences do allow a gap in x immediately followed by a gap in y , and vice versa. For example,

x : ATGTTCAA--TCT

y : ATGATC--CCTCT

Problem 4. (25 points)

For all subproblems assume a regular scoring scheme with parameters m , $-s$ and $-d$.

(a) (10 points) Bacterial DNA is often organized into circular molecules. Given two strings x and y of length n and m , respectively, there are n circular shifts of x , and m circular shifts of y ; therefore, there are nm pairs of circular shifts. Build an efficient algorithm to find the best global alignment among the nm different pairs of circular shifts. The trivial $O(n^2 m^2)$ algorithm is not good enough. A cubic time solution (i.e. $O(nm \cdot \min(n, m))$) will receive half credit. Anything faster will receive full credit.

(b) (8 points) Let P be a pattern of length n , and T be a text of length m . The *tandem repeat problem* is to find an interval in T that has the best global alignment with some tandem repeat of P . Let P^m be the concatenation of P with itself m times (the maximum number of times the tandem repeat could possibly appear in T). The tandem repeat problem is equivalent to computing the local alignment between P^m and T , and the standard local alignment algorithm solves this problem in $O(nm^2)$ time. Find an approach that solves the tandem repeat problem in $O(nm)$ time.

This particular problem arises in studying the secondary structure of proteins that form what is called a *coiled coil*. In that context, P represents a *motif* or *domain* (a pattern for our purposes) that can repeat in the protein an unknown number of times, and T represents the protein.

(c) (7 points) Since the biological significance of the optimal alignment is sometimes uncertain, and optimality depends on the choice of (often disputed) weights, it is useful to efficiently produce or study a set of *suboptimal* (but close) alignments in addition to the optimal one. Given two strings X and Y (of lengths n and m) and a parameter δ , show how to construct the following matrix in $O(nm)$ time: $M(i,j) = 1$ if and only if there is an alignment of X and Y in which characters $X[i]$ and $Y[j]$ are aligned with each other and the value of the alignment is within δ of the maximum value alignment of X and Y. That is, if $F(n,m)$ is the value of the optimal alignment, then the best alignment that puts $X[i]$ opposite $Y[j]$ should have value at least $F(n,m) - \delta$.