Algorithms for Molecular Biology
© Ron Shamir
School of Computer Science
Tel Aviv University
November 25, 2001

Algorithms for Molecular Biology: Assignment No. 2
due on December 20th

Credit: Solve 13 items out of 16 for 26/75.

1. **Finding Repeats**: In the problem of local alignment between two different sequences A and B, one has to find a pair of subsequences, one in A and the other in B, with maximum similarity. Suppose that we want to find a pair of subsequences *within* a sequence A with maximum similarity (it is also called the "optimal inexact repeat"). Argue why we cannot simply compute the local alignment between A and itself. Give an algorithm to solve the optimal inexact repeat problem for the following cases. Analyze the space complexity and running time of your solution.

   (a) The two subsequences may overlap.

   (b) Overlap between the subsequences is *not* allowed.

   (c) The two subsequences must be adjacent in A, that is, $A = uw_lw_rv$ and w_l, w_r have the highest similarity over all such subsequences of A. This type of repeat is called **a tandem repeat** and it occurs very often in biological sequences.

   (d) The same as the previous item, but we require that w_l and $(w_r^c)^R$ have the highest edit similarity over all such subsequences of A, where $(A^c)^R$ is the reverse complement of a DNA sequence A. This biological phenomenon is called an *inverted repeat*.

2. A is called a non-contiguous *supersequence* of B if B is a non-contiguous subsequence of A. Give a polynomial algorithm to find a *shortest* non-contiguous supersequence of both B_1 and B_2. Prove and analyze it.

3. Show how Hirschberg's linear space technique can be used for the computation of a local alignment, and for the computation of a global alignment with an affine gap penalty (i.e. $g(k) = \alpha + k \times \beta$).

4. **RNA Secondary Structure:** Recall that mRNA is a single-stranded molecule. mRNA tends to fold in a way that can be represented in two dimensions, with the strand twisting, but never crossing itself. In such a folding, called *secondary structure*,

pairs of non consecutive nucleotides A-U, or G-C can form hydrogen bonds along the "inner" side of the strand. This usually leads to structural elements called *hairpins* (on account of physical resemblance), *loops*, and *bulges* (see figure 1). Observe that all such pairings are either disjoint or nested, i.e., if s_i pairs with $s_{i'}$ and s_j pairs with $s_{j'}$, then $i < j < j'$ implies $i' < j$ or $j' < i'$. Such a set of hydrogen bond pairings that can be mutually formed is called a *configuration*.

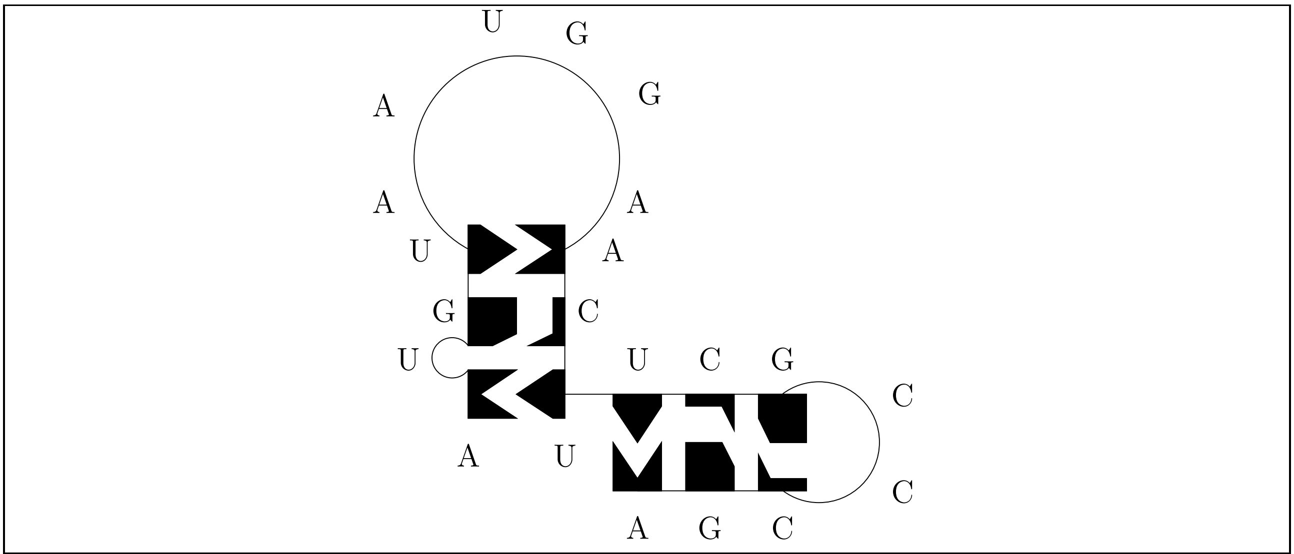

Figure 1: A possible secondary structure for the RNA sequence `AUGUAAUGGAACUUCGCCCGA`. The thin solid line is the mRNA strand. The `U` in the second position forms a bulge of size 1, while the terminating `UCGCCCGA` sequence forms a hairpin with the loop `CC` of size 2.

How many possible configurations are there? Give a dynamic programming algorithm, that given some mRNA sequence finds a configuration with a maximal number of pairings.

5. **Arbitrary Gap Penalties:** Suppose a gap penalty $f(k)$ is charged for a gap of length k. In addition, a matrix of similarity values $\sigma(x, y)$ is given.

   (a) Give (and analyze) a dynamic programming algorithm to compute the optimal global alignment.

   (b) Can you do better for convex f?

6. **Sequence Comparison with at most l Mismatches**. Many heuristics for rapid similarity searches use the following idea to find sequences with "enough" similarity. It is based on the following lemma: Suppose that $a_1, \ldots, a_t$ and $b_1, \ldots, b_t$ can be aligned with at most l mismatches. Then: (i) For $k \le \lfloor \frac{t}{l+1} \rfloor$, $a_1, \ldots, a_t$ and $b_1, \ldots, b_t$ share at least $t - (l+1)k + 1$ k-tuples. (ii) For $k = \lfloor \frac{t}{l+1} \rfloor$, $a_1, \ldots, a_t$ and $b_1, \ldots, b_t$ share at least

one k-tuple (a k-tuple is a subsequence of length k which occurs in both sequences). Prove the lemma, and show how it can be used to rapidly search all sequences in a database which can be aligned with a given query sequence with at most l mismatches. This idea is called *filtration*.

7. **Alignment Score and Probability:** Assume the following model for mutations:

   - Insertions/deletions never happen.
   - During each infinitesimally small period of time Δt, in each sequence position a substitution occurs with probability $3\alpha\Delta t$, replacing the original nucleotide with some other nucleotide, which is chosen uniformly.
   - All sequence positions are independent.

   (a) Prove that according to this model, if a position is occupied by a nucleotide x at time $t = 0$, then the probability that this position is occupied by a nucleotide y at time $t = T$, is:
   $$p_{xy} = \frac{1}{4} + (\delta_{xy} - \frac{1}{4})e^{-4\alpha T}$$
   with $\delta_{xy} = \begin{matrix} 1 & x = y \\ 0 & \text{otherwise} \end{matrix}$

   (b) Suppose some known sequence S is left to evolve for a period of time T according to the above model. For any sequence S', calculate the probability that this is the sequence resulting from the experiment. Show the connection between this probability and an appropriate ungapped alignment score.

8. **Speedup for Edit-Distance Alignment:** Consider a $t \times t$ sub-rectangle R, in the $(n+1) \times (n+1)$ matrix M. The *value* of R is its t^2 entries. Denote by $|R$ and by $R|$ its leftmost and rightmost columns respectively, and by $\overline{R}$ ($\underline{R}$) its top (bottom) row. In the *edit distance* version of the alignment problem, indel operations and mismatches have each value $+1$, while a match has value 0. Let M be the $(n+1) \times (n+1)$ dynamic programming for computing the edit distance between two n-long sequences over a finite alphabet.

   (a) Prove that there are $O(nK^t)$ possible values of R, for some fixed K. Hint: Prove that $M_{i,j} - M_{i+1,j} \in \{0, \pm 1\}$, and obtain a bound on the number of possible values for $|R$,$\overline{R}$.

   (b) Assume a memory of size $O(n^2)$ of unbounded integers, each of which is randomly accessible in one time unit. Show how this idea allows an $O(\frac{n^2}{\log^2 n})$ time algorithm for computing the edit-distance in this model of computation.

9. The gene contents of the mouse and human genome are very similar, and genes appearing together as a group in one species, tend to appear together in the other as well. This correspondence can be represented by *comparative maps* as follows: Let $(h_1, \ldots, h_n)$ be the list of human genes, ordered according to their occurrence along a human chromosome, where for each h_i there is a mouse homologue h'_i. A *human-mouse comparative map* is a list $(m_1, \ldots, m_n)$ where h'_i is on the mouse chromosome m_i. In general, a continuous fragment along the human chromosome maps to the same mouse chromosome, but there are some disruptions. Define a *labeling* of a map as a partition of $(m_1, \ldots, m_n)$ into continuous fragments: $((m_{L(1)}, \ldots, m_{R(1)}), (m_{L(2)}, \ldots, m_{R(2)}), \ldots, (m_{L(k)}, \ldots, m_{R(k)}))$, with the fragment $(m_{L(j)}, \ldots, m_{R(j)})$ being labeled with a mouse chromosome c_j. Typically we would like most of $(m_{L(j)}, \ldots, m_{R(j)})$ to be c_j, but some errors should be tolerated. For constants α, β, define the *cost* of a labeling to be $\alpha k + \beta \sum_{j=1}^{k} |\{L(j) \leq i \leq R(j) | m_i \neq c_j\}|$. α is a cost per label, and β is a cost per disruption of a label. Give an algorithm to compute the minimum-cost labeling.

10. **Perfect matches in the genome:** In a random genome of length 3×10^9, how many hits are expected to perfectly match a query sequence of length 15? Browse `http://www.ncbi.nlm.nih.gov/blast/`. Choose blastn (first available link) and search for an arbitrary 15-long sequence. Check the box for *Perform ungapped alignment* and in the text-area type two lines: the first consisting only of the "greater than" (>) sign, and the second containing your sequence (what is your sequence?). When you view the results, scroll down to the 'Alignments' section, and look at each of the alignments themselves. How many perfect matches did you find? What is the total size of the database searched? Had this database been a single, long, random sequence (of the same size), compute (without relying on the search output) how many perfect matches should have we expected to observe?