

Introduction to Bioinformatics Algorithms

Homework 4

Saad Mneimneh, Computer Science, Hunter College of CUNY

Problem 1: Spliced alignments

(a) Consider the Exon Chaining problem in the case where all intervals have the same weight. For this setting, the best chain is obviously the one that has the maximum number of non-overlapping intervals. Describe a **greedy** algorithm that finds the optimal solution.

(b) For the Spliced Alignment problem presented in the book, if m is the length of the target t , n is the length of the genome g , and l is the sum of all block lengths, then the algorithm runs in $O(ml|V| + |V|^2)$, where $O(|V|^2)$ is needed to determine the order of blocks using topological sort ($|V|$ is the number of blocks). If $l = O(n|V|)$, this algorithm runs in $O(mn|V|^2)$ time.

Suggest a way to improve this running time by scanning all blocks from left to right. In particular, if $S(i, B, j)$ represents the score of the optimal spliced alignment that ends in g_i and t_j where $i \in B$, compute S like this:

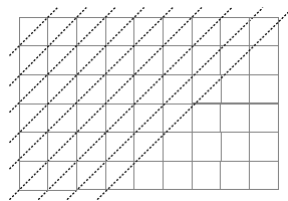
```
for each  $(i, B)$  in lexicographic order
  for each  $j \leftarrow 0$  to  $m$ 
    do ...
```

When $(i, B, j) = (\text{right}(B), B, j)$, we are “ending” block B . So after $S(\text{right}(B), B, j)$ is computed, update a quantity $W(j)$ if that’s the highest seen for j among all block endings so far. This quantity will be used to avoid going through the list of all blocks that precede a given one (see recurrences in the book).

What is the running time of your algorithm?

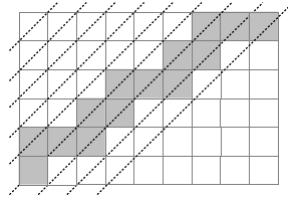
Problem 2: Space efficient balanced alignment

In parallel implementations of alignment, most algorithms fill the table diagonal-wise because the entries required for filling the diagonal are all contained in the previous two diagonals, and there are no intra-dependencies within a diagonal, so each diagonal can be computed in parallel.



(a) Implement a linear-space (no backtracking) global alignment algorithm that computes the optimal score and works diagonally.

(b) Modify your algorithm to save a set of entries (i, j) such that $ij = (m - i)(n - j)$, roughly, so that every global alignment must cross this set.



(c) In the linear-space approach to sequence alignment, the original problem of size $m \times n$ is reduced to two subproblems of sizes $\frac{m}{2}j$ and $\frac{m}{2}(n - j)$. In a fast parallel implementation of sequence alignment, it is desirable to have a *balanced partition* that breaks the original problem into subproblems of equal sizes. Design a space-efficient version with balanced partitioning (but you don't have to implement it).

Hint: The optimal alignment must pass through one of the entries you saved.

Problem 3: Partial digest

Rewrite the pseudocode for the “practical” partial digest algorithm using fewer lines, by moving any change to X and L into the recursive call. In addition, make it stop when a solution is found. Implement the algorithm using the vector library in C++.