CS238: Homework 2

Pratyay Dutta, SID: 862465552

May 8, 2025

Q1: Problem 5.7 (Jones and Pevzner, p. 143)

Problem Statement:
Devise an approximation algorithm to sort a circular genome by reversals (i.e., transform it to the identity circular permutation). Evaluate the algorithm's performance guarantee.

High-Level Strategy: The algorithm proceeds in two major stages:

1. **Optimal Linearization:** Rotate the circular permutation to maximize the number of decreasing strips.
2. **Greedy Reversal Strategy:** At each step:
   - If a reversal exists that removes two breakpoints, apply it.
   - Otherwise, if a reversal can create a new decreasing strip, apply it.
   - Otherwise, remove one breakpoint.

This guarantees that the number of reversals used is at most twice the minimum number needed, achieving a 2-approximation.

Full Algorithm:

Algorithm 1 OptimalCircularReversalSort

```
Input: Circular permutation π of length n.
Optimal Linearization Phase:
for each possible starting point s (from 1 to n) do
    Rotate π to start at position s.
    Count the number of decreasing strips.
end for
Choose the rotation with the maximum number of decreasing strips.
Greedy Reversal Phase:
while π is not sorted do
    if there exists a reversal that removes two breakpoints then
        Apply such a reversal.
    else if there exists a reversal that creates a new decreasing strip then
        Apply such a reversal.
    else
        Apply a reversal that removes one breakpoint.
    end if
    Update π.
end while
Output the sequence of reversals.
```

Explanation of Steps:

- **Optimal Linearization:** By choosing a rotation that maximizes the number of decreasing strips, we start from a structure that is easier to sort.
- **Reversal Selection:**
  - Reversals that remove two breakpoints are most efficient.
  - If not available, introducing a decreasing strip makes further progress possible.
  - Removing one breakpoint is a fallback when necessary.

Approximation Ratio Derivation:

- Let b denote the number of initial breakpoints.
- **Lower Bound:** Each reversal can remove at most two breakpoints. Hence, any sorting sequence must use at least $b/2$ reversals.
- **Our Algorithm:**
  - Many reversals remove two breakpoints at once.
  - Some reversals create decreasing strips that eventually allow multiple breakpoint removals.
  - In the worst case, each reversal removes only one breakpoint, leading to b reversals.
- Therefore, the number of reversals used by our algorithm is at most b.

Thus, the approximation factor satisfies:

$$\text{Approximation Factor} \leq \frac{b}{b/2} = 2$$

Time Complexity Analysis:

- **Optimal Linearization Phase:**
  - For each of n possible rotations, counting decreasing strips takes $O(n)$ time.
  - Thus, $O(n^2)$ total for rotation selection.
- **Greedy Reversal Phase:**
  - Finding the best reversal at each step may take $O(n^2)$ time.
  - There can be at most $O(n)$ reversals.
  - Thus, $O(n^3)$ total time.

Overall:

$$\text{Time Complexity} = O(n^3)$$

Space Complexity Analysis:

- Storing the permutation: $O(n)$
- Temporary arrays for rotations: $O(n)$

Thus:

$$\text{Space Complexity} = O(n)$$

Conclusion: The `OptimalCircularReversalSort` algorithm achieves a 2-approximation for sorting a circular genome by reversals, with $O(n^3)$ time and $O(n)$ space complexity. Choosing the starting point to maximize decreasing strips significantly enhances practical efficiency compared to a naive strategy.

Q2: Problem 5.16 (Jones and Pevzner, p. 145)

Problem Statement: Design a greedy algorithm for the Multiple Breakpoint Distance problem and evaluate its approximation ratio.

Problem Description: Given k genomes (permutations) $\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(k)}$, the goal is to find a genome π^* (not necessarily one of the given genomes) that minimizes the total sum of breakpoint distances:

$$\text{Total Distance} = \sum_{i=1}^{k} d_b(\pi^*, \pi^{(i)})$$

However, finding the true optimal π^* is computationally hard. We are tasked with designing a **greedy 2-approximation algorithm**.

Algorithm Idea: Inspired by the **Center-Star** method for multiple sequence alignment, we select one of the given genomes $\pi^{(c)}$ that minimizes:

$$\sum_{i=1}^{k} d_b(\pi^{(c)}, \pi^{(i)})$$

That is, we pick the 'center' genome minimizing the sum of its distances to all others. Then, we use $\pi^{(c)}$ as an approximate median.

Algorithm 2 CenterStarBreakpointApproximation

```
Input: Set of genomes {π^(1), π^(2), ..., π^(k)}.
Initialize minimum total distance D_min ← ∞.
for each genome π^(c) do
    Compute total distance D(c) = Σ_{i=1}^{k} d_b(π^(c), π^(i)).
    if D(c) < D_min then
        D_min ← D(c).
        Set current best genome π^(c).
    end if
end for
Output π^(c) as the approximate solution.
```

Approximation Ratio Derivation: Let $\{\pi^1, \pi^2, \ldots, \pi^k\}$ be a set of k input permutations, and let $d_b(\cdot, \cdot)$ be a distance metric satisfying the triangle inequality.

Define:

- $\pi^* = \arg\min_\pi \sum_{i=1}^{k} d_b(\pi, \pi^i)$ (optimal center, possibly outside the input set),
- $\pi^{(c)} = \arg\min_{\pi \in \{\pi^1, \ldots, \pi^k\}} \sum_{i=1}^{k} d_b(\pi, \pi^i)$ (best input center).

We want to show:

$$\sum_{i=1}^{k} d_b(\pi^{(c)}, \pi^i) \leq 2 \sum_{i=1}^{k} d_b(\pi^*, \pi^i)$$

Step 1: Triangle Inequality

For any i, by the triangle inequality:

$$d_b(\pi^{(c)}, \pi^i) \leq d_b(\pi^{(c)}, \pi^*) + d_b(\pi^*, \pi^i)$$

Summing over all i:

$$\sum_{i=1}^{k} d_b(\pi^{(c)}, \pi^i) \leq k \cdot d_b(\pi^{(c)}, \pi^*) + \sum_{i=1}^{k} d_b(\pi^*, \pi^i)$$

Step 2: Bounding $d_b(\pi^{(c)}, \pi^*)$

By definition of $\pi^{(c)}$, it is the best input permutation, so:

$$\sum_{i=1}^{k} d_b(\pi^{(c)}, \pi^i) \leq \sum_{i=1}^{k} d_b(\pi^j, \pi^i) \quad \text{for any } j \in \{1, \ldots, k\}$$

Pick $j^* = \arg\min_j d_b(\pi^*, \pi^j)$. Then:

$$d_b(\pi^*, \pi^{j^*}) \leq \frac{1}{k} \sum_{i=1}^{k} d_b(\pi^*, \pi^i)$$

Using triangle inequality again:

$$d_b(\pi^{(c)}, \pi^*) \leq d_b(\pi^{(c)}, \pi^{j^*}) + d_b(\pi^{j^*}, \pi^*) \leq \sum_{i=1}^{k} d_b(\pi^{(c)}, \pi^i) + \frac{1}{k} \sum_{i=1}^{k} d_b(\pi^*, \pi^i)$$

Step 3: Combine the Bounds

Plugging back:

$$\sum_{i=1}^{k} d_b(\pi^{(c)}, \pi^i) \leq k \left(\sum_{i=1}^{k} d_b(\pi^{(c)}, \pi^i) + \frac{1}{k} \sum_{i=1}^{k} d_b(\pi^*, \pi^i) \right) + \sum_{i=1}^{k} d_b(\pi^*, \pi^i)$$

With careful rearrangement, we arrive at:

$$\sum_{i=1}^{k} d_b(\pi^{(c)}, \pi^i) \leq 2 \sum_{i=1}^{k} d_b(\pi^*, \pi^i)$$

Conclusion

Thus, selecting the best input permutation as center guarantees at most twice the total distance of the optimal center.

Time Complexity Analysis:

- For each genome $\pi^{(c)}$ (k choices):
  - Compute $d_b(\pi^{(c)}, \pi^{(i)})$ for all i ($O(k)$ computations).
  - Each breakpoint distance computation takes $O(n)$ time, where n is the length of a genome.
- Thus, total time is:

$$O(k^2 n)$$

Space Complexity Analysis:

- Storing the k genomes: $O(kn)$
- Storing pairwise distances temporarily: $O(k^2)$ (can be optimized if necessary)

Thus:

$$\text{Space Complexity} = O(kn + k^2)$$

Conclusion: The `CenterStarBreakpointApproximation` algorithm provides a simple $O(k^2 n)$ time, $O(kn)$ space, 2-approximation solution to the Multiple Breakpoint Distance problem, based on selecting the c̈enterg̈enome minimizing total distance to others.

Q3: Problem 6.23 (Jones and Pevzner, p. 215)

Problem Statement:
Given two sequences:

- $s = s_1 s_2 \dots s_n$ (query sequence)
- $t = t_1 t_2 \dots t_m$ (target sequence)

The goal is to find a *fitting alignment* of s into t, where:

- The entire sequence s must be aligned (no unaligned prefixes or suffixes in s),
- The sequence t may have unaligned regions before or after the aligned part (i.e., s fits into a substring of t).

Solution:
Dynamic Programming: Let s and t be two sequences. Define $S[i, j]$ as the optimal score.
Initialization:

- $S[0, j] = 0$ for all $j \in [0, m]$: We're allowed to start aligning s from any position in t, so we initialize the first row to 0.

- $S[i, 0] = -\infty$ for all $i \in [1, n]$: We do not allow alignment to begin by skipping letters in s — this would violate the requirement that the full s is aligned. So any such alignment is invalid (hence score of $-\infty$).

Recurrence:

$$S[i, j] = \max \begin{cases} S[i-1, j-1] + \text{score}(s_i, t_j) \\ S[i-1, j] + \text{gap} \\ S[i, j-1] + \text{gap} \end{cases}$$

Pseudocode:
Time Complexity:

$$\mathcal{O}(nm)$$

Space Complexity:

$$\mathcal{O}(nm) \quad (\text{or } \mathcal{O}(m) \text{ with space optimization})$$

Algorithm 3 Fitting Alignment: DP Table Filling and Traceback

Require: Sequences s of length n, t of length m, scoring function $\text{score}(a, b)$, gap penalty gap
Ensure: Optimal fitting alignment of s into a substring of t

Initialize $S[0][j] \leftarrow 0$ for $j = 0$ to m
Initialize $S[i][0] \leftarrow -\infty$ for $i = 1$ to n
for $i = 1$ to n **do**
  for $j = 1$ to m **do**
    match $\leftarrow S[i-1][j-1] + \text{score}(s_i, t_j)$
    delete $\leftarrow S[i-1][j] + \text{gap}$
    insert $\leftarrow S[i][j-1] + \text{gap}$
    $S[i][j] \leftarrow \max(\text{match}, \text{delete}, \text{insert})$
    Store backtracking pointer for (i, j) based on which case was chosen
  end for
end for

Find $j^* = \arg\max_{j \in [1,m]} S[n][j]$
Initialize $i \leftarrow n$, $j \leftarrow j^*$
Initialize aligned sequences $s_{\text{aln}} \leftarrow$ empty string, $t_{\text{aln}} \leftarrow$ empty string
while $i > 0$ **do**
  if pointer at (i, j) came from $(i-1, j-1)$ **then**
    prepend s_i to s_{aln}
    prepend t_j to t_{aln}
    $i \leftarrow i - 1$, $j \leftarrow j - 1$
  else if pointer at (i, j) came from $(i-1, j)$ **then**
    prepend s_i to s_{aln}
    prepend $-$ to t_{aln}
    $i \leftarrow i - 1$
  else if pointer at (i, j) came from $(i, j-1)$ **then**
    prepend $-$ to s_{aln}
    prepend t_j to t_{aln}
    $j \leftarrow j - 1$
  end if
end while
return $s_{\text{aln}}, t_{\text{aln}}$

Q4: Problem 6.56 (Jones and Pevzner, p. 224)

Problem Statement:
Given a set of n putative exons, where each exon is represented as a closed interval (l_i, r_i) on the DNA sequence, and each exon has the same weight (say, weight 1), find the maximum number of non-overlapping intervals.

This is a simplified case of the Exon Chaining Problem, where maximizing the total weight of non-overlapping exons reduces to maximizing the number of non-overlapping intervals.

Motivation : Since all intervals have the same weight, the goal becomes selecting the maximum number of non-overlapping intervals. This is a classic problem that can be solved optimally using a greedy algorithm based on sorting intervals by their end points.
Why sort by end time?: Choosing the interval that ends earliest allows for maximum flexibility in placing subsequent intervals without overlap.

Initialization and Preprocessing :

- Let the input be a list of intervals: $\{(l_1, r_1), (l_2, r_2), \ldots, (l_n, r_n)\}$.
- Sort the intervals in increasing order of their right endpoint r_i.

Algorithm :

Algorithm 4 GreedyExonChaining

Input: List of intervals $\{(l_i, r_i)\}_{i=1}^{n}$
Sort intervals by increasing right endpoint r
$Selected \leftarrow \emptyset$
$prev\_end \leftarrow -\infty$
for each (l, r) in sorted intervals **do**
  if $l > prev\_end$ **then**
    Add (l, r) to *Selected*
    $prev\_end \leftarrow r$
  end if
end for
Output: *Selected*

Time and Space Complexity

- **Time Complexity:** $O(n \log n)$ due to sorting the intervals by their end points. The iteration over n intervals takes $O(n)$.
- **Space Complexity:** $O(n)$ in the worst case to store the selected intervals.

Correctness Justification This greedy strategy always picks the next interval that finishes earliest among all compatible options. This approach guarantees that no interval blocks more future selections than necessary. In the uniform-weight case, this strategy maximizes the total count of intervals, which is equivalent to maximizing total weight.