Advanced Algorithms - Final Assignment

Vinh Ngo

April 2025

1 Problem Description

Problem. Given an unknown string $T = t_1 t_2 \dots t_m$ of length m, and n noisy copies $S_1, S_2, \dots, S_n$ of T. Each noisy copy is generated by processing the original string T character by character, where at each position i (for $1 \leq i \leq m$), exactly one of the following mutually exclusive operations occurs independently with the specified probability:

- With probability r: The character t_i is deleted
- With probability r: A random character is inserted before t_i, then t_i is kept
- With probability r: The character t_i is replaced with a different character
- With probability $1 - 3r$: The character t_i is kept unchanged

Note that these errors cause the noisy copies to have varying lengths with characters from the original sequence potentially appearing at different positions. The goal is to reconstruct the original string T from the noisy copies $S_1, S_2, \dots, S_n$.

2 Center-Star Multiple Sequence Alignment Algorithm

2.1 Algorithm Overview

The approach to reconstructing the original string T follows a two-phase strategy: first, we align all noisy copies using Multiple Sequence Alignment (MSA), and then apply majority voting on each column to determine the most likely character at each position of the original sequence.

2.2 The Center-Star Heuristic

Multiple Sequence Alignment is known to be NP-hard for more than two sequences [3], making optimal alignment computationally infeasible for this problem. Instead, we employ the Center-Star heuristic, which reduces the complexity to polynomial time by selecting one sequence as the "center" and aligning all other sequences to it.
The Center-Star method provides a $2(1-\frac{1}{n})$-approximation to the optimal alignment [1], which is suitable for the reconstruction task while maintaining reasonable computational efficiency.

2.3 Algorithm Description

The implementation follows three main steps:

2.3.1 Step 1: Pairwise Alignment and Center Sequence Selection

We first compute all $\binom{n}{2} = \frac{n(n-1)}{2}$ pairwise alignments using the Needleman-Wunsch [2] algorithm (`nw.py`). This dynamic programming approach finds optimal global alignments between each pair of sequences with time complexity $O(m^2)$ per alignment, where m is the average sequence length.
The center sequence is selected as the one with the maximum sum of alignment scores with all other sequences:

$$\mathit{center} = \arg\max_{i \in \{1,\dots,n\}} \sum_{j=1, j \neq i}^{n} \text{score}(S_i, S_j) \tag{1}$$

This heuristic attempts to find the sequence most similar to all others, which is likely to be close to the true sequence.

2.3.2 Step 2: Progressive Alignment

Using the center sequence as a reference, we build a multiple sequence alignment incrementally. When adding each new sequence, we:

- Retrieve its pairwise alignment with the center sequence
- Adjust gap positions to maintain consistency with the existing MSA
- Add the aligned sequence to the MSA

The `align_similar` and `adjust` functions in the pseudo-code handle gap insertion to maintain alignment.

2.3.3 Step 3: Consensus Generation/Sequence Reconstruction

Finally, we analyze each column of the MSA to determine the most likely original character:

- For each column, we count the occurrences of each character
- The most frequent non-gap character is selected for the consensus
- This majority voting approach helps correct errors present in individual sequences

The consensus sequence represents my best estimate of the original sequence T.
The complete implementation in `reconstruction.py` includes both the algorithm and evaluation framework, allowing empirical comparison with the theoretical success probability.

2.3.4 Pseudo-code

Algorithm 1: Helper Functions

```
1 Procedure AlignSimilar($s1$, $s2$)
      // Takes two pre-aligned sequences and identifies positions requiring
         additional gaps
      // Returns indices where gaps need to be inserted in each sequence to make
         them structurally identical
      // Works by comparing characters position by position and inserting gaps
         where needed
2     return change_indices1, change_indices2
3 Procedure AdjustWithGaps($string\_list$, $indices$)
      // Takes a list of strings and a list of indices
      // Inserts a gap character ('-') at each specified index in all strings in
         the list
      // Used to maintain alignment consistency across all sequences in the MSA
4 Procedure NeedlemanWunsch($A$, $B$, match, mismatch, gap)
      // Dynamic programming algorithm for optimal global sequence alignment
      // Creates a scoring matrix based on match/mismatch rewards and gap penalties
      // Fills matrix by considering match/mismatch, insertion, or deletion at each
         cell
      // Traces back through matrix to construct optimal alignment
5     return aligned sequences and alignment score
```

Algorithm 2: Center-Star Multiple Sequence Alignment

```
Input: Set of sequences $S = \{S_1, S_2, \ldots, S_n\}$, scoring parameters (match, mismatch, gap)
Output: Multiple sequence alignment of all sequences in $S$
Procedure CenterStarMSA($S$, match, mismatch, gap)
    // Step 1: Calculate pairwise alignments and find the center
    $scores \leftarrow [0] \times n$ // Stores total alignment score for each sequence
    $alignments \leftarrow$ empty $n \times n$ matrix
    for each pair $(i, j)$ where $1 \leq i < j \leq n$ do
        // Align each pair of sequences using Needleman-Wunsch
        $(S'_i, S'_j, score) \leftarrow$NeedlemanWunsch($S_i, S_j$, match, mismatch, gap)
        $alignments[i][j] \leftarrow (S'_i, S'_j)$ // Store aligned sequences
        $alignments[j][i] \leftarrow (S'_j, S'_i)$ // Store symmetric alignments
        $scores[i] \leftarrow scores[i] + score$ // Accumulate scores for center determination
        $scores[j] \leftarrow scores[j] + score$
    end
    $center \leftarrow$ index of maximum value in $scores$ // Select center sequence

    // Step 2: Progressive alignment using the center sequence
    $MSA \leftarrow []$ // Will store all aligned sequences
    for each sequence $S_i$ in $S$ do
        if $i = center$ then
            continue // Skip the center sequence itself
        end
        if $MSA$ is empty then
            // Initialize MSA with first aligned pair
            $MSA$.append($alignments[center][i][0]$) // Center sequence aligned to $S_i$
            $MSA$.append($alignments[center][i][1]$) // $S_i$ aligned to center
        end
        else
            // Align new sequence to existing MSA
            $new \leftarrow [alignments[center][i][0], alignments[center][i][1]]$
            $(changes1, changes2) \leftarrow$ AlignSimilar(MSA[0], new[0])
            // Insert gaps to maintain alignment
            AdjustWithGaps($MSA$, $changes1$) // Add gaps to MSA sequences
            AdjustWithGaps($new$, $changes2$) // Add gaps to new sequences
            // Add new aligned sequence to MSA
            $MSA$.append($new[1]$)
        end
    end
    return $MSA$
```

Algorithm 3: Sequence Reconstruction from MSA

```
Input: Multiple sequence alignment $MSA$
Output: Reconstructed sequence $\hat{T}$
Procedure ReconstructSequence($MSA$)
    $consensus \leftarrow []$
    $msa\_length \leftarrow |MSA[0]|$
    for $i \leftarrow 0$ to $msa\_length - 1$ do
        // Extract column $i$ from all sequences in the MSA
        $column \leftarrow [seq[i]$ for $seq$ in $MSA]$
        // Count occurrences of each character in the column
        $counts \leftarrow \text{Counter}(column)$
        // Find most common characters
        $most\_common \leftarrow \text{counts.most\_common}(2)$
        // Add most frequent character to consensus if it's not a gap
        if $most\_common$ and $most\_common[0][0] \neq$ '-' then
            $consensus.\text{append}(most\_common[0][0])$
        end
    end
    return join($consensus$)
```

3 Theoretical Analysis

3.1 Time Complexity Analysis

The algorithm has the time complexity $O(n^2 \cdot m^2)$, where:

- n is the number of noisy sequences
- m is the expected average length of sequences
- Computing all $\binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$ pairwise alignments
- Each Needleman-Wunsch alignment requires $O(m^2)$ time due to the dynamic programming matrix

3.2 Probabilistic Success Analysis

We analyze the probability of successfully reconstructing the original sequence T using the Center-Star MSA algorithm followed by majority consensus. The analysis is based on certain simplifying assumptions to make the probabilistic calculation tractable.

Theorem 3.1 (Position-wise Success Probability)**.** Given a true sequence T of length m, error rate r per position (where $0 \leq 3r < \frac{1}{2}$), and n noisy samples, the probability of correctly reconstructing character t_i at position i is:
$P(\hat{t}_i = t_i) = \sum_{k=\lfloor \frac{n}{2} \rfloor+1}^{n} \binom{n}{k} p_i^k (1-p_i)^{n-k}$
where p_i is the probability that a noisy copy has the correct character at position i after alignment.

Proof. For this analysis, we make the assumption that the Center-Star MSA algorithm correctly aligns homologous positions - that is, characters derived from the same position in the original sequence are aligned in the same column of the MSA. This is a simplifying assumption as the Center-Star is a heuristic that does not guarantee optimal alignment.
Under this assumption, for each position i in the original sequence T, each sample independently has probability $p_i = 1 - 3r$ of containing the correct character at this position after alignment.
Let X_i be the random variable representing the number of samples that have the correct character at position i in the MSA. This follows a binomial distribution: $X_i \sim \text{Binomial}(n, p_i)$.
For the consensus to yield the correct character at position i, we need a majority of sequences to have the correct character: $P(\hat{t}_i = t_i) = P(X_i > \frac{n}{2}) = \sum_{k=\lfloor \frac{n}{2} \rfloor+1}^{n} \binom{n}{k} p_i^k (1-p_i)^{n-k}$

Theorem 3.2 (Overall Success Probability)**.** The probability of correctly reconstructing the entire sequence T is:
$P(\hat{T} = T) = \prod_{i=1}^{m} P(\hat{t}_i = t_i)$
Assuming independence between positions after correct alignment.

Remark (The "Correct Alignment" Assumption)**.** The Center-Star algorithm selects one sequence as the center and aligns all other sequences to it. The probability analysis assumes this alignment correctly matches homologous positions. This assumption requires clarification:

- The choice of center sequence significantly impacts alignment quality. Ideally, the center should be the sequence most similar to the true sequence.
- In practice, the center is selected as the sequence with the minimum sum of distances to all other sequences, which approximates but may not identify the optimal center.
- The assumption becomes less valid as:
  - The error rate r increases
  - The sequence length m increases
  - The number of samples n decreases

When this assumption is violated, the actual success probability will be lower than my theoretical bound.

Remark (The Algorithm is Monte Carlo)**.** The Center-Star MSA with majority consensus is a Monte Carlo algorithm, with the following characteristics:

- It always returns an answer (a reconstructed sequence) in polynomial time
- The success probability depends on the error rate r and sample size n
- We can increase the success probability by using more samples

4 Implementation and Evaluation

To validate the theoretical analysis and measure the practical performance of the Center-Star MSA algorithm, I implemented the solution and conducted some experimental evaluation. The source code is available at: `https://github.com/vinhqngo5/center-star-msa-reconstruction`.

4.1 Implementation Overview

The repository is structured as follows:

- `cstar/`: Main implementation directory
  - `nw.py`: Implementation of Needleman-Wunsch algorithm for pairwise sequence alignment
  - `reconstruction.py`: Main script implementing Center-Star MSA and evaluation framework

The implementation follows the approach described in the algorithmic section. The `NeedlemanWunsch` class performs pairwise alignments, the `CenterStar` class implements the center selection and progressive alignment. Theoretical success probability calculation is included to compare empirical results with the mathematical model.

4.2 Experimental Results

A representative experiment with moderate error rate and a decent sample size is shown below:

```
$ python3 cstar/reconstruction.py --generate --true_length 100 --error_rate 0.05 \
                                  --num_samples 50 --num_runs 10
```

The experiment generates 10 random true sequences of length 100, creates 50 noisy copies of each with error rate $r = 0.05$, and attempts to reconstruct the original sequence using the proposed approach.

```
$ python3 cstar/reconstruction.py --generate --true_length 100 --error_rate 0.05 \
                                  --num_samples 50 --num_runs 10
--- Starting Evaluation ---
Parameters: True Length=100, r=0.05, Num Samples=50, Num Runs=10

[...]

--- Run 10/10 ---
  True Sequence (first 50 chars): CCACAACTAAGAATCCATTATTGCGAGTCTGAGTCCTTTCTGTTTCGACT...
  Generated 50 noisy samples.
  Calculating 1225 pairwise alignments...
  Finding center sequence... (Index: 46, Score: 3201)
  Center sequence length: 102
  > Edit distances of center to true sequence: 22
  Center sequence: CCACAACTAAGATCCATTATTTGCGATCTGAGTCCTTTCTGTTTCGACTT...
  Aligning all sequences to the center sequence...
  MSA complete. Aligned 50 sequences.
  Generating consensus sequence from MSA...
  Reconstruction Time: 0.88 seconds
  Estimated Sequence Length: 100
  Estimated Sequence (first 50 chars): CCACAACTAAGAATCCATTATTGCGAGTCTGAGTCCTTTCTGTTTCGACT...
  Result: SUCCESS

--- Evaluation Summary ---
Parameters: True Length=100, r=0.05, Num Samples=50, Num Runs=10
Success Rate: 80.00% (8/10)
Average Edit Distance (when failed): 1.00
Average Reconstruction Time: 0.83 seconds

--- Theoretical vs Empirical Comparison ---
Theoretical success probability: 100.00%
Empirical success rate: 80.00%
ANALYSIS: Empirical success rate is relatively close to the theoretical prediction.
```

References

[1] Dan Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology*, 55(1):141–154, 1993.

[2] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[3] LUSHENG WANG and TAO JIANG. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994. PMID: 8790475.