Bioinformatics

-

Generalization of Dynamic Programming Techniques for Multiple Sequence Alignment

Name : Pablo Deputter
Student ID : s0205440
e-mail : pablo.deputter@student.uantwerpen.be

June 21, 2024

1 Introduction

In this project, I implemented a generalized dynamic programming algorithm for multiple sequence alignment (MSA), extending the Needleman-Wunsch and Smith-Waterman algorithms designed for pairwise alignment. This MSA algorithm was then also applied to a biological problem.

The implementation was successful and produced results consistent with the provided test files. The algorithm can be executed in two modes: command-line interface (CLI) and as a standalone script. The CLI mode can be initiated with the command `python main.py`, while the standalone script can be run by commenting out the CLI part in the `main.py` file and uncommenting the MSA functions. The CLI can utilize default scoring parameters or custom settings specified in a configuration file. The default settings are detailed in the `data/settings.json` file, following the parameters instructed in the assignment. Further instructions for the CLI can be found be found in the `README.md` file or by running `python main.py -h`.

Additionally, the project includes several test files, both self-created and provided with the assignment, located in the testing directory. A script to run these tests is available and can be executed using `python -m unittest tests/testMSA.py`. Parameters such as scoring weights can be adjusted through the settings file or by supplying a different file in the CLI. The project accepts FASTA files as input and outputs the alignment in a specified format.

1.1 Project Structure

The project is organized as follows:

- **Data:** Contains input data files, output results, and the configuration file (`data/settings.json`) for scoring parameters (match, mismatch, gap penalties).
- **report.pdf:** The main report document.
- **requirements.txt:** Lists project dependencies.
- **src:** Holds the source code for the project, including the main script (`src/msa.py`) and other utility functions.
- **tests:** Contains the test cases for the project.
  - **data/expected:** Contains the expected output for the tests.
  - **data/input:** Contains the input data for the tests.
  - **test_msa.py:** Includes the tests for the MSA algorithm.
- **main.py:** The primary script for running the CLI. This file can be modified to run the MSA algorithm without the CLI by (un)commenting appropriate sections of code.

2 Generalization of Needleman-Wunsch Algorithm & Smith-Waterman to k Sequences

The generalization of the Needleman-Wunsch and Smith-Waterman algorithms to k sequences involves extending the dynamic programming approach used in pairwise alignments to handle multiple dimensions. This section outlines the methodology and key components of this generalization in detail.

The dynamic programming (DP) framework for multiple sequence alignment (MSA) is based on constructing a multi-dimensional scoring matrix. For k sequences, a k-dimensional matrix $F(i_1, i_2, \ldots, i_k)$ is used, where each dimension corresponds to one of the sequences. The size of the matrix is determined by the lengths of the sequences plus one to accommodate initial gap penalties. Specifically, if the sequences have lengths $n_1, n_2, \ldots, n_k$, the matrix will have dimensions $(n_1 + 1) \times (n_2 + 1) \times \ldots \times (n_k + 1)$.

2.1 Initialization

The initialization of the scoring matrix involves setting the scores for the initial rows and columns to represent gap penalties for the sequences. For example, for sequences s_1, s_2, s_3 with lengths $2, 2, 2$

respectively, the dimensions of the matrix would be $(3, 3, 3)$. The initialized matrix for global alignment with a gap penalty of -4 would look like this:

$$\left[\begin{bmatrix} 0 & -4 & -8 \\ -4 & -8 & -12 \\ -8 & -12 & -16 \end{bmatrix}, \begin{bmatrix} -4 & -8 & -12 \\ -8 & -12 & -16 \\ -12 & -16 & -20 \end{bmatrix}, \begin{bmatrix} -8 & -12 & -16 \\ -12 & -16 & -20 \\ -16 & -20 & -24 \end{bmatrix}\right]$$

So the value of a specific cell in the matrix using global alignment is calculated as the sum of the indices multiplied by the gap penalty: $F(i_1, i_2, \ldots, i_k) = \sum_{j=1}^{k} i_j \cdot \text{gap\_penalty}$

For local alignment, the initial scores are set to zero to allow alignment to start from any point within the sequences. Thus all values in the matrix are initialized to zero: $F(i_1, i_2, \ldots, i_k) = 0$

2.2 Scoring Scheme

The scoring scheme for MSA extends the pairwise alignment scores by considering all possible pairwise combinations of sequences. Given sequences $s_1, s_2, \ldots, s_k$, the score for aligning positions in these sequences is calculated as the sum of the pairwise comparison scores.

For pairwise alignment, the recurrence relations are given by:

$$F(i, j) = \max \begin{pmatrix} F(i-1, j-1) + s(c_i, c_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \\ 0 \text{ (when local)} \end{pmatrix}$$

To generalize this to k sequences, we create a k-dimensional matrix. The recurrence relation for the score $F(i_1, i_2, \ldots, i_k)$ is:

$$F(i_1, i_2, \ldots, i_k) = \max \left\{ \begin{array}{l} 0, \quad \text{(when local)} \\ F(i_1 - 1, i_2 - 1, \ldots, i_k - 1) + \sum_{1 \le p < q \le k} s(c_p, c_q), \\ F(i_1, i_2 - 1, \ldots, i_k - 1) + \sum_{1 \le p < q \le k} s(c_p, c_q) - d, \\ F(i_1 - 1, i_2, \ldots, i_k - 1) + \sum_{1 \le p < q \le k} s(c_p, c_q) - d, \\ \vdots \\ F(i_1, i_2, \ldots, i_k - 1) - kd \end{array} \right\}$$

Here, $F(i_1, i_2, \ldots, i_k)$ represents the score at position $(i_1, i_2, \ldots, i_k)$ in the matrix, $s(c_p, c_q)$ is the match or mismatch score between characters c_p and c_q from sequences s_p and s_q, and d is the gap penalty.

2.3 Matrix Filling Procedure

The matrix filling procedure involves iterating through each cell of the k-dimensional matrix and calculating the optimal score for each position based on possible predecessor cells. The number of possible movements or predecessors for each cell is $2^k - 1$, where k is the number of sequences.

For each cell $F(i_1, i_2, \ldots, i_k)$, the score is computed as the maximum of the possible scores obtained by aligning the sequences in various ways. This includes extending alignments by introducing gaps in different sequences or by aligning characters directly. The calculation for each cell involves the same recurrence relation described earlier:

$$F(i_1, i_2, \ldots, i_k) = \max_{\Delta_1, \ldots, \Delta_k \in \{0,1\}, \Delta_1 + \ldots + \Delta_k \neq 0} \left(F(i_1 - \Delta_1, i_2 - \Delta_2, \ldots, i_k - \Delta_k) + \sum_{1 \le p < q \le k} s(c_p, c_q) \right)$$

Here, $\Delta_1, \ldots, \Delta_k$ represent the possible movements (0 for no movement, 1 for a step forward) for each sequence, and the condition $\Delta_1 + \ldots + \Delta_k \neq 0$ ensures that at least one sequence moves in each step. The term $F(i_1 - \Delta_1, i_2 - \Delta_2, \ldots, i_k - \Delta_k)$ represents the score of the predecessor cell from which the current cell could be reached by the respective movements. The sum $\sum_{1 \le p < q \le k} s(c_p, c_q)$ represents the sum of the pairwise comparison scores between the characters c_p and c_q.

2.4 Traceback for Alignment Reconstruction

Once the scoring matrix is filled, the optimal alignment is reconstructed through a traceback procedure. Starting from the cell with the highest score (for local alignment) or the cell corresponding to the full lengths of all sequences (for global alignment), the algorithm traces back through the matrix to determine the optimal alignment path. The traceback involves moving through the matrix by following the path that maximizes the alignment score.

The final aligned sequences are obtained by concatenating the characters and gaps encountered during the traceback. I did not keep a separate traceback matrix, but instead dynamically computed the predecessors during the traceback process to optimize memory usage.

2.5 Algorithmic Complexity

2.5.1 Space Complexity

The space complexity is determined by the size of the k-dimensional scoring matrix. For k sequences of lengths $n_1, n_2, \ldots, n_k$, the matrix requires:

$$O(n_1 \cdot n_2 \cdot \ldots \cdot n_k) = O(n^k) \quad \text{if } n \text{ is the maximum sequence length}$$

This space complexity arises because we need to store a score for each possible alignment of the sequences, resulting in a matrix with n^k cells.

2.5.2 Time Complexity

The time complexity can be broken down into several parts: initialization, matrix filling, and traceback. Each of these steps contributes to the overall complexity of the algorithm.

The initialization step involves setting the initial scores. This requires $O(n^k)$ time since each cell in the k-dimensional matrix must be initialized, resulting in n^k operations.

The matrix filling step is the most computationally intensive part of the algorithm. For each cell $F(i_1, i_2, \ldots, i_k)$, the score is computed by considering all possible predecessors as described earlier. Here is the detailed breakdown:

1. **Nodes to Compute**: The number of cells to fill is $O(n^k)$ since each of the k sequences can have up to n positions.
2. **Predecessors per Node**: Each cell has $2^k - 1$ predecessors. This is because each sequence can either stay at its current position or move to the next position, resulting in 2^k combinations minus the all-zero case (where no sequence moves).
3. **Pairwise Comparisons per Node**: For each cell, the score is computed by summing the pairwise comparison scores of characters from the k sequences. This requires $O(k^2)$ comparisons.

Thus, the total time complexity for filling the matrix is:

$$O(n^k \cdot (2^k - 1) \cdot k^2) = O((2n)^k \cdot k^2)$$

The traceback procedure is linear in the number of cells visited, which is at most $O(n^k)$. However, since the traceback involves only one path, its complexity is significantly lower than the matrix filling step. Thus the overall time complexity of the algorithm is dominated by the matrix filling step: $O((2n)^k \cdot k^2)$.

3 Application on Biological Problem

In this part of the project, I applied the generalized dynamic programming algorithm for multiple sequence alignment to a biological problem involving T cell receptor sequences.

T cells are part of the adaptive immune system and are responsible for recognizing pathogens. Two protein chains of the T cell receptor, the T cell receptor alpha (TCA) chain and T cell receptor beta (TCB) chain, are involved in recognition. Both chains have a J region, which has multiple versions with small variations. The provided sequences are:

```
>unknown_J_region_1
GYSSASKIIFGSGTRLSIRP

>unknown_J_region_2
NTEAFFGQGTRLTVV

>unknown_J_region_3
NYGYTFGSGTRLTVV
```

3.1 Global MSA

When globally aligning the sequences using the default scoring parameters and my implementation, the following alignment was obtained:

```
unknown_J_region_1: GYSSASKIIFGSGTRLSIRP
unknown_J_region_2: NTE.AF...FGQGTRLTVV.
unknown_J_region_3: NYG.YT...FGSGTRLTVV.
Alignment score: 45
```

From this alignment, it is clear that sequences `unknown_J_region_2` and `unknown_J_region_3` align more closely with each other, suggesting they are both TCB J regions. This is due to the higher similarity and shorter length typical of the TCB J region sequences. The sequence `unknown_J_region_1` aligns less closely with the other two and is longer, which is characteristic of the TCA J region, indicating it is likely the TCA J region.

3.2 Local MSA

When performing local alignment with the default scoring parameters, but now with a mismatch penalty of -4 instead of the default mismatch penalty, the following alignment was obtained:

```
unknown_J_region_1: FGSGTRL
unknown_J_region_2: FGQGTRL
unknown_J_region_3: FGSGTRL
Alignment score: 87
```

The local alignment reveals a conserved region in all three sequences: `FG[SQ]GTRL` in `unknown_J_region_1` and `unknown_J_region_3`, and `FGQGTRL` in `unknown_J_region_2`. This conserved region suggests a functional importance in the recognition process of the T cell receptor.

The decision to use a higher mismatch penalty of -4 in the local MSA was motivated by the need to emphasize the importance of exact matches over mismatches. By increasing the mismatch penalty, the alignment algorithm is encouraged to find regions with high conservation and penalize regions with mismatches more heavily. This helps in identifying truly conserved regions that are likely to be functionally important.