GENETIC SEQUENCE ALIGNMENT TOOL IN MATLAB

Genetic Sequence Alignment Tool in MATLAB Using Needleman-Wunsch and Smith-Waterman Algorithms

Parth Chopra

Nipissing University

Table of Contents

1.1 Abstract

This project introduces a MATLAB-based graphical user interface (GUI) designed for pairwise genetic sequence alignment. The tool incorporates two key alignment algorithms—Needleman-Wunsch (for global alignment) and Smith-Waterman (for local alignment). Users can load DNA sequences in FASTA format, perform alignments, and visualize results for further analysis. The tool is aimed at facilitating sequence similarity analysis, which is essential in bioinformatics applications such as evolutionary analysis, gene prediction, and protein structure modelling.

1.2 Introduction

1.2.1 DNA and RNA

Biological sequences such as DNA and RNA are composed of long chains of molecules called nucleotides. DNA (Deoxyribonucleic Acid) consists of four types of nucleotides represented by the letters A (adenine), T (thymine), C (cytosine), and G (guanine). RNA (Ribonucleic Acid) is similar but has uracil (U) instead of thymine. These sequences store genetic information and guide processes like protein synthesis.

In base pairing—how sequences align with each other—adenine (A) always pairs with thymine (T) in DNA, or with uracil (U) in RNA, while cytosine (C) pairs with guanine (G). For example, a DNA strand with the sequence **A-T-C-G** would align with a complementary DNA strand **T-A-G-C**, or with an RNA strand **U-A-G-C**. This base-pairing rule is essential for processes like replication and transcription, and it's also the foundation for sequence alignment techniques used in bioinformatics.

1.2.2 What is sequence alignment and why is it important?

Sequence alignment is a fundamental technique in **bioinformatics** used to compare two or more DNA, RNA, or protein sequences to identify regions of **similarity**. These similarities may indicate the sequences' **functional, structural, or evolutionary relationships**. At its core, sequence alignment involves arranging sequences in such a way that:

- **Identical or similar characters** (nucleotides or amino acids) are aligned together.
- **Gaps (insertions or deletions)** are introduced where needed to maximize the alignment score.

1.2.3 Global vs local alignment.

There are two main methods of DNA sequencing:

1. ***Global Alignment*** (e.g., Needleman-Wunsch):

- Aligns sequences **end-to-end**.
- Useful when sequences are of **similar length** and generally related throughout.

2. ***Local Alignment*** (e.g., Smith-Waterman):

    - Aligns the **most similar subsequences**.
    - Ideal when sequences have **conserved regions** within longer or unrelated sequences.

1.2.4 Significance of Needleman-Wunsch and Smith-Waterman.

Both the **Needleman-Wunsch** and **Smith-Waterman** algorithms are foundational in bioinformatics and computational biology, and understanding their principles is crucial to understanding how sequence comparisons work.

The Needleman-Wunsch algorithm, a global alignment algorithm, is designed for aligning **entire sequences** from beginning to end. It uses a dynamic programming matrix to find the highest-scoring alignment, considering all positions. It compares genes or proteins that are **closely related** and expected to be similar along their **entire length**. This algorithm is widely used in **phylogenetic analysis** and **whole-genome comparisons**. This algorithm laid the groundwork for sequence alignment algorithms. This **optimal** global alignment and demonstrates the **value of dynamic programming** in bioinformatics.

The **Smith-Waterman Algorithm,** a local alignment algorithm targets the most similar subregions within two sequences. This algorithm also uses dynamic programming, but resets negative scores to zero to focus only on high-scoring local regions to find the common subregions. It is used to compared protein and nucleotide domains or functional motifs across unrelated proteins and nucleotides. It is useful for identifying **conserved regions** such as enzyme active sites. It Allows for flexible comparison when sequences differ greatly in size or contain non-homologous regions. This is a foundational tool for **database searching** (e.g., in BLAST before heuristics were introduced).

1.2.5 Motivation for building this tool.

The motivation for developing this sequence alignment tool stems from both educational and practical needs within the bioinformatics community:

1. **Educational Value**:
- Facilitates the visualization of how alignment algorithms operate at a granular level.
- Enhances understanding of the differences between global and local alignments through interactive demonstrations.
2. **User Accessibility**:

- Many existing tools (e.g., EMBOSS) can be challenging for beginners, particularly those without extensive programming experience.
- A GUI-based tool simplifies the alignment process, making it accessible to a wider audience, including biologists and researchers.

3. **Visualization and Real-time Feedback**:

- Provides visual representation of the alignment matrix, helping users understand the impact of matches, mismatches, and gaps on alignment quality.
- Real-time updates enhance user comprehension, particularly in educational contexts.

4. **Flexibility for Research and Experimentation**:

- Offers users the opportunity to customize parameters (e.g., gap penalties, scoring matrices), fostering experimentation.
- Can be extended to include more advanced features, such as multiple sequence alignment or scoring heatmaps.

5. **Foundation for Further Development**:

- Serves as a foundation for more complex bioinformatic tools, such as multiple sequence alignment engines, phylogenetic tree construction, and comparative genomics platforms.

1.3 Features and Functionality

- FASTA Loading
- Algorithm Choice
- Alignment Execution
- Save Output
- Visual Feedback

1.4 Methodology

1.4.1 FASTA File Parsing

The FASTA format is a universally accepted text-based standard for representing nucleotide or protein sequences. It begins with a header line, denoted by a ">" character, followed by one or more lines containing the sequence itself. In this study, gene sequences relevant to disease-linked genes were obtained in FASTA format from NCBI and other genomic databases. To facilitate downstream analysis, a custom script was implemented to parse and extract raw sequence data. This function, written in MATLAB, reads the file line by line, discards the header, and concatenates the remaining lines into a continuous sequence string. This streamlined approach enables efficient handling of FASTA data for alignment, comparison, and further bioinformatic analysis.

1.4.2 The Needleman-Wunsch (NW) algorithm: The Global Sequence Alignment

The Needleman-Wunsch (NW) algorithm is a foundational dynamic programming technique designed for **global sequence alignment**. It ensures the optimal alignment of two biological sequences—typically proteins or nucleotides—by comparing them from beginning to end. The algorithm operates under the assumption that the entire length of both sequences should be aligned, making it ideal for sequences of roughly equal length with a high degree of similarity.

1.4.2.1 Implementation Details

The algorithm uses a **scoring matrix** to systematically compute alignment scores. This matrix is initialized with gap penalties in the first row and column, reflecting the cost of aligning a sequence with initial gaps. Each cell in the matrix, $F(i,j)$, is filled using the recurrence relation:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + \text{match/mismatch score}, \\ F(i-1,j) + \text{gap penalty}, \\ F(i,j-1) + \text{gap penalty} \end{cases}$$

After populating the matrix, a **traceback** is performed starting from the bottom-right cell to reconstruct the optimal alignment path.

1.4.2.2 Time Complexity

The time and space complexity of the Needleman-Wunsch algorithm is `O(mn)`, where `m` and `n` are the lengths of the input sequences. This makes it computationally intensive for very long sequences but effective for high-fidelity global alignments.

1.4.2.3 Pseudocode

```
def needleman_wunsch(seq1, seq2, match_score=1, mismatch_score=-1, gap_penalty=-2):
    m = len(seq1) + 1
    n = len(seq2) + 1

    # Initialize scoring matrix
    matrix = [[0 for _ in range(n)] for _ in range(m)]

    # Initialize first row and column with gap penalties
    for i in range(m):
        matrix[i][0] = i * gap_penalty
    for j in range(n):
```

```
        matrix[0][j] = j * gap_penalty


    # Fill scoring matrix
    for i in range(1, m):
        for j in range(1, n):
            match = matrix[i-1][j-1] + (match_score if seq1[i-1] == seq2[j-1] else
mismatch_score)
            delete = matrix[i-1][j] + gap_penalty
            insert = matrix[i][j-1] + gap_penalty
            matrix[i][j] = max(match, delete, insert)


    # Traceback to get aligned sequences
    aligned_seq1 = ""
    aligned_seq2 = ""
    i, j = m - 1, n - 1


    while i > 0 and j > 0:
        score = matrix[i][j]
        diag = matrix[i-1][j-1]
        up = matrix[i-1][j]
        left = matrix[i][j-1]


        if score == diag + (match_score if seq1[i-1] == seq2[j-1] else
mismatch_score):
            aligned_seq1 = seq1[i-1] + aligned_seq1
            aligned_seq2 = seq2[j-1] + aligned_seq2
            i -= 1
            j -= 1
        elif score == up + gap_penalty:
            aligned_seq1 = seq1[i-1] + aligned_seq1
            aligned_seq2 = "-" + aligned_seq2
            i -= 1
        else:
            aligned_seq1 = "-" + aligned_seq1
            aligned_seq2 = seq2[j-1] + aligned_seq2
            j -= 1
```

```
    # Fill remaining gaps
    while i > 0:
        aligned_seq1 = seq1[i-1] + aligned_seq1
        aligned_seq2 = "-" + aligned_seq2
        i -= 1
    while j > 0:
        aligned_seq1 = "-" + aligned_seq1
        aligned_seq2 = seq2[j-1] + aligned_seq2
        j -= 1

    return aligned_seq1, aligned_seq2
```

1.4.3 Smith-Waterman Algorithm: Local Sequence Alignment

The Smith-Waterman (SW) algorithm builds on the dynamic programming principles of Needleman-Wunsch but is designed for **local alignment**. Unlike global alignment, local alignment focuses on aligning the most similar subregions between two sequences. This is particularly useful when the sequences differ significantly in length or share only conserved motifs.

1.4.3.1 Implementation

The primary modification in Smith-Waterman is the introduction of **zero as a lower bound** in the matrix-filling step:

$$F(i,j) = \max \begin{cases} 0, \\ F(i-1,j-1) + \text{match/mismatch score}, \\ F(i-1,j) + \text{gap penalty}, \\ F(i,j-1) + \text{gap penalty} \end{cases}$$

This zero-value ensures that negative scores are avoided, effectively restarting the alignment when dissimilar regions are encountered.

The **traceback** process begins at the cell with the **maximum score** in the matrix and proceeds until a cell with a score of zero is reached. This results in an alignment of only the most homologous subsequences.

1.4.3.2 Time Complexity

Like Needleman-Wunsch, the Smith-Waterman algorithm has a time complexity of $O(mn)$ but with the added advantage of focusing computation on locally significant regions, thereby providing biologically meaningful results even in divergent sequences.

1.4.3.3 Pseudocode

```
def smith_waterman(seq1, seq2, match_score=1, mismatch_score=-1, gap_penalty=-2):
    m = len(seq1) + 1
    n = len(seq2) + 1

    # Initialize scoring matrix with 0s
    matrix = [[0 for _ in range(n)] for _ in range(m)]

    max_score = 0
    max_pos = (0, 0)

    # Fill scoring matrix
    for i in range(1, m):
        for j in range(1, n):
            match = matrix[i-1][j-1] + (match_score if seq1[i-1] == seq2[j-1] else
mismatch_score)
            delete = matrix[i-1][j] + gap_penalty
            insert = matrix[i][j-1] + gap_penalty
            matrix[i][j] = max(0, match, delete, insert)

            if matrix[i][j] > max_score:
                max_score = matrix[i][j]
                max_pos = (i, j)

    # Traceback from highest scoring cell
    aligned_seq1 = ""
    aligned_seq2 = ""
    i, j = max_pos

    while i > 0 and j > 0 and matrix[i][j] != 0:
        score = matrix[i][j]
        diag = matrix[i-1][j-1]
        up = matrix[i-1][j]
        left = matrix[i][j-1]
```

```
        if score == diag + (match_score if seq1[i-1] == seq2[j-1] else
mismatch_score):
            aligned_seq1 = seq1[i-1] + aligned_seq1
            aligned_seq2 = seq2[j-1] + aligned_seq2
            i -= 1
            j -= 1
        elif score == up + gap_penalty:
            aligned_seq1 = seq1[i-1] + aligned_seq1
            aligned_seq2 = "-" + aligned_seq2
            i -= 1
        elif score == left + gap_penalty:
            aligned_seq1 = "-" + aligned_seq1
            aligned_seq2 = seq2[j-1] + aligned_seq2
            j -= 1


    return aligned_seq1, aligned_seq2
```

1.4.4 Graphical User Interface (GUI) Implementation

To enhance accessibility and user experience, a custom **Graphical User Interface (GUI)** was developed using MATLAB. The GUI facilitates seamless interaction with the alignment tool without requiring command-line expertise.

1.4.4.1 Interface Components

The interface consists of the following elements:

- **Buttons** for uploading sequence files (FASTA format).
- **Dropdown menus** to select alignment algorithms (Needleman-Wunsch or Smith-Waterman).
- **Text areas** for displaying input sequences and alignment results.
- **Score settings** for match, mismatch, and gap penalties.
- **Save and Export** options for storing alignment results.

1.4.4.2 Execution Flow

The GUI follows a structured workflow:

1. **Input**: Users upload two sequences via the file selector.
2. **Configuration**: Algorithm and scoring parameters are selected.

3. **Alignment**: Upon clicking “Align,” the selected algorithm computes the optimal alignment using the defined parameters.
4. **Output**: The aligned sequences are displayed, and users can optionally save the results.

1.5 Screenshots

1.5.1 GUI Interface

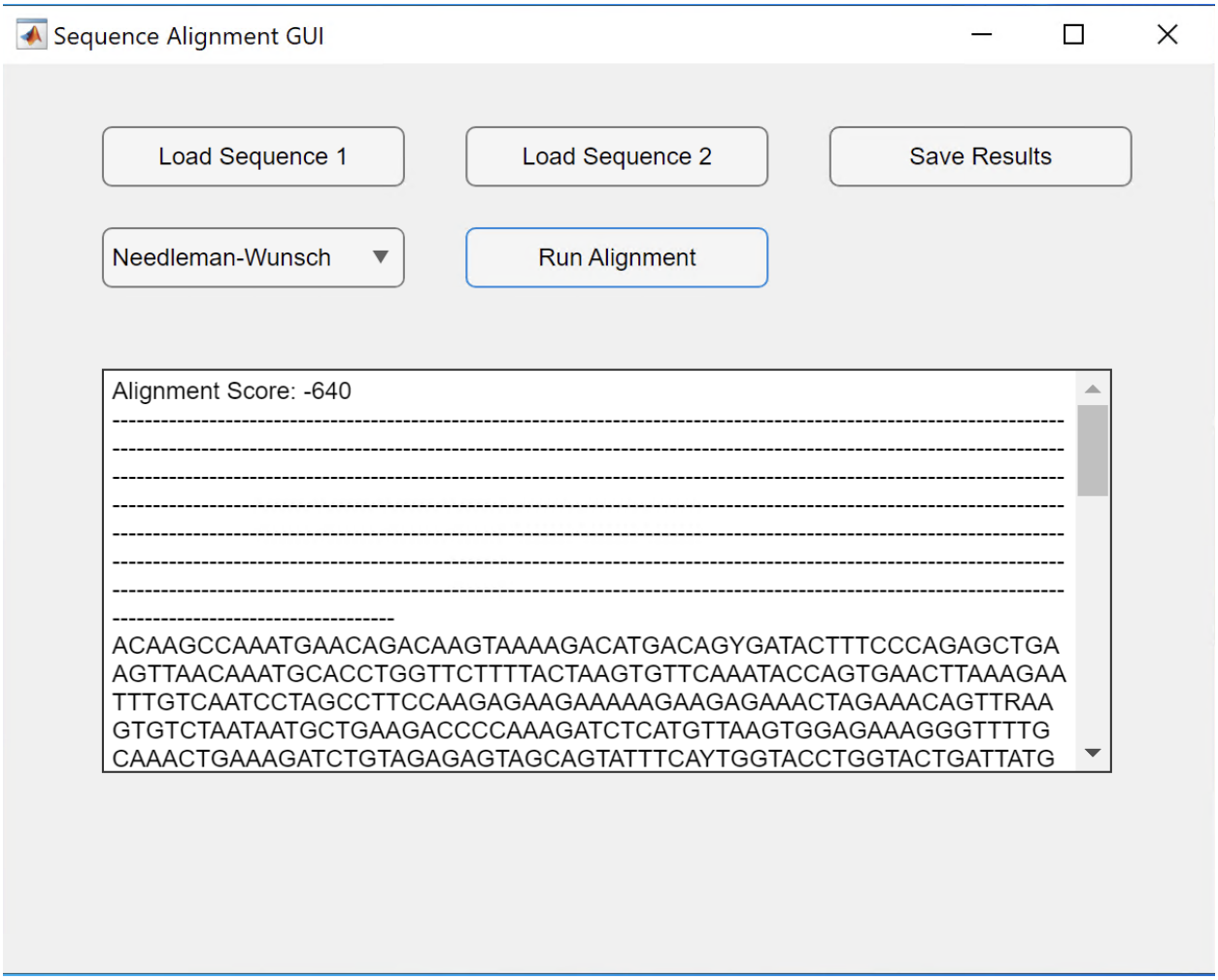

Figure 1 Output area with aligned sequences

Figure 2 Needleman-Wunsch Algorithm output.

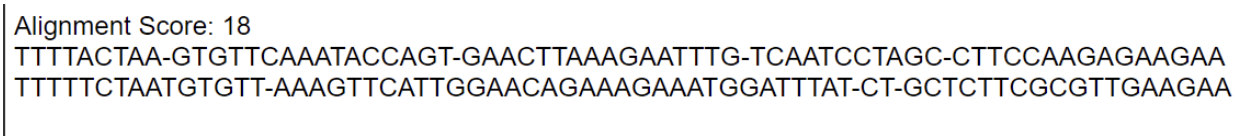

Figure 3 Smith-Waterman Algorithm output.

1.5.2 File selection dialogue

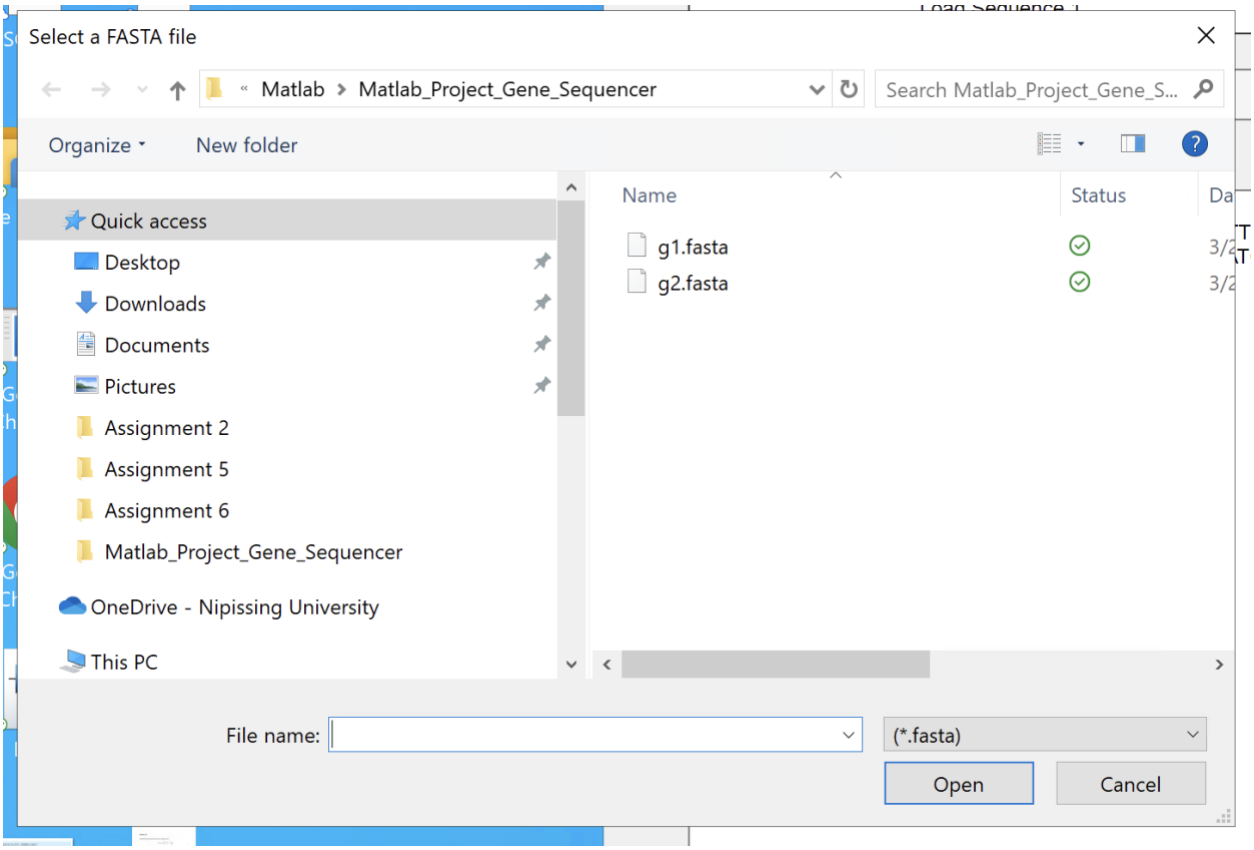

Figure 4 Choose and save FASTA files with GUI.

1.6 Folder Structure

```
Directory of C:\Users\pchopra725\OneDrive - Nipissing University\Matlab\Matlab_Project_Gene_Sequencer

4/18/2025  08:52 PM    <DIR>          .
4/18/2025  08:52 PM    <DIR>          ..
4/02/2025  10:02 PM             3,902 alignment_gui_1.asv
4/02/2025  10:03 PM             4,002 alignment_gui_1.m
4/03/2025  10:07 AM               188 alignment_results.txt
3/24/2025  02:59 PM               906 g1.fasta
3/24/2025  02:56 PM             1,480 g2.fasta
3/24/2025  03:29 PM             1,813 needleman_wunsch.m
3/24/2025  03:30 PM               459 read_fasta.m
3/24/2025  05:12 PM             2,020 smith_waterman.m
3/24/2025  05:41 PM               506 Trial1.m
3/24/2025  04:44 PM           (5,096) Y3GKMVKE114_search_strategy.asn
              10 File(s)         20,372 bytes
               2 Dir(s)  642,334,789,632 bytes free
```

1.7 Usage Instructions

1. Run alignment_gui_1.m in MATLAB.

2. Load two FASTA files.
3. Choose an alignment algorithm.
4. Click “Run Alignment.”
5. View results in the text area.
6. Click “Save Results” to export

1.8 References

Needleman, S.B. and Wunsch, C.D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3), pp.443–453. doi:https://doi.org/10.1016/0022-2836(70)90057-4.

Smith, T.F. and Waterman, M.S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1), pp.195–197. doi:https://doi.org/10.1016/0022-2836(81)90087-5.

File- g1.fasta - https://www.ncbi.nlm.nih.gov/nuccore/MW716260.1

File- g2.fasta- https://www.ncbi.nlm.nih.gov/nuccore/U61268.1

For more FASTA files visit - https://www.ncbi.nlm.nih.gov/nuccore