CSCI 6470 Project – Spring 2025

School of Computing

March 16, 2025

1 Project Description (10% of overall grade)

This project is to design a dynamic programming algorithm to accomplish the task of **Pairwise Alignment**.

1. Input: two DNA sequences x and y consisting of letters `A. C. G, T`, a scoring function $\sigma(l_1, l_2)$, where l_1 and l_2 are letters taken from $\{\texttt{A.C.G}, \texttt{T}\}$ and penalty score δ for gap "_";

2. Output: an alignment between sequences x and y such that the column-wise total score $f(i, j)$, where $i = |x|$ and $j = |y|$, is maximized, where every column is scored with either $S(l_1, l_2)$ between the two aligned letters l_1 and l_2 in the column or with δ if either letter is "_".

2 Requirements

Develop the algorithm into a program (in Python, Java, C++, or any other programming language that does not have built-in pairwise alignment functions). The score function σ and gap penalty δ will be given to you during demo. You may modify a scoring scheme used in **Edit Distance** or **Longest Common Subsequence** to suit your needs in debugging your program.

3 Bonus (additional 5% of overall grade)

Modify your program to accommodate the *affine gap penalty*. .

That is, if there are m consecutive gaps in an alignment, e.g.,

```
A G C G A A T G T
  G C G _ _ _ _ _
```

the penalty for the 5 gaps is not $5 \times \delta$. Instead, it is a linear function $\alpha + \beta(m-1)$, where α is the *gap opening penalty* and β is the *gap extension penalty*. To deal with this type of gap penalty, you have two choices to modify your algorithm.

3.1 Option 1

This is to "guess" how long the streak of gaps may be when there is a gap to consider.

$$f(i,j) = \max \begin{cases} f(i-1,j-1) + S(x[i], y[j]) & \\ f(i,j-k) + \alpha + \beta(k-1) & k = 1, \ldots, |y| \\ f(i-k,j) + \alpha + \beta(k-1) & k = 1, \ldots, |x| \end{cases}$$

Note that this formulation of f will result in $O(n^3)$ time complexity instead of $O(n^2)$. **Why?**

3.2 Option 2

To avoid the $O(n^3)$ time complexity, your algorithm can be designed to remember three different states M, I and D to score an alignment. Transitions from M to the other two states, transitions from I to I and M, and transitions from D to D and M are allowed. Correspondingly, three alignment functions $f_M(i,j)$, $f_I(i,j)$, and $f_D(i,j)$ are needed, where

$$f_M(i,j) = \max \begin{cases} f_M(i-1,j-1) + S(x[i], y[j]) \\ f_I(i-1,j-1) + S(x[i], y[j]) \\ f_D(i-1,j-1) + S(x[i], y[j]) \end{cases}$$

and $f_I(i,j)$, and $f_D(i,j)$ can be defined accordingly. **How?**

This option keeps the time complexity $O(n^2)$.

4 Grading

The project will be graded based on in-person demo in the instructor's office. Due before the last day of classes (April 27, 2025).