CSCI 4470 Algorithms, Fall 2019: Project

Due on November 22, 2019

Dr. Liming Cai

Kang Tze, Ng
811506330

Introduction

This project presents a dynamic programming solution to the **Pairwise Alignment** problem, which is an extension to the MAXIMUM SEQUENCE MATCHING problem presented in homework assignment #4. The following report presents (1) a description of the problem with examples, followed by (2) the solution in pseudocode.

1 Problem Description

The Pairwise Alignment problem is an extension of the MAXIMUM SEQUENCE MATCHING problem, which is in turn is an extension to the LONGEST COMMON SUBSEQUENCE problem. We are given an input of two (DNA) sequences and a scoring scheme. The algorithm provided must obtain a highest scoring alignment of the two sequences and the optimal score of the abovementioned alignment. For example, between two sequences `relation` and `reality`, the following matching

```
re_lation
realit_y_
```

has the total score

$$\mu(r,r) + \mu(e,e) + \mu(\_,a) + \mu(l,l) + \mu(a,i) + \mu(t,t) + \mu(i,) + \mu(o,y) + \mu(n,\_)$$

where a gap is denoted by the _ symbol. For the purpose of this project, only DNA sequences composed of the symbols `A`, `G`, `C` and `T` must be matched.

1.1 Theoretical Viewpoint

Given two sequences, it is clear to see that an exhaustive enumeration of all possible alignments must provide an optimal solution. However, such an algorithm, while simple to produce, has an impractical time complexity. Therefore, the solution to such a problem is a dynamic programming algorithm.

2 Implementation

2.1 Objective Function

For any two subsequences **x** and **y** of length m and n, let $S(m,n)$ be the objective function on these sequences denoted by $x_1x_2\ldots x_m$ and $y_1y_2\ldots y_n$. Then, the objective function S to obtain the optimal solution can be described by

$$S(i,j) = max \begin{cases} S(i-1,j) + \mu(x_i,\_) \\ S(i,j-1) + \mu(\_,y_j) \\ S(i-1,j-1) + \mu(x_i,y_j) \end{cases}$$

where the base cases are

$$\begin{aligned} S(0,0) &= 0 \\ S(i,-1) &= -\infty \qquad 1 < i < m \\ S(-1,j) &= -\infty \qquad 1 < j < n. \end{aligned}$$

The objective function uses the **backtracking** property. Each instance of the objective function for two subsequences of x and y only uses the most optimal subsequence immediately before it.

2.2 Algorithm

We first construct two $m+1$ by $n+1$ matrices, one to keep track of the score of each instance of the objective function *(score)*, and one to keep track of which prior objective function that score came from in order to trace the sequences afterwards *(trace)*. Then, we populate the first row and column of both matrices as follows:

0	-6	$\dots$	-24	-30
-6				
$\dots$		$\dots$		
-24				
-30		$\dots$		

Table 1: *score* matrix

0	$\leftarrow$	$\dots$	$\leftarrow$	$\leftarrow$
$\uparrow$				
$\dots$		$\dots$		
$\uparrow$				
$\uparrow$		$\dots$		

Table 2: *trace* matrix

starting from row 0, we want a downwards move from the $i-1th$ to the ith row in column 0 to correspond to matching the x_ith symbol to a gap. Similarly, a rightwards move along row 0 should correspond to matching the y_jth symbol to a gap. In this case, we assume that the gap penalty is uniformly -6 for all symbols. Then, using the objective function, we populate each remaining entry according to the objective function. Upon completion, the optimal score should be obtained by reading the entry in the **bottom right corner** of the *score* matrix. The path taken can then be obtained by tracing the arrows from the bottom right corner to the top left corner of the *trace* matrix.

2.3 Example

Under the following scheme:

	A	C	G	T	$\_$
A	5	-2	-2	-2	-6
C	-2	5	-2	-2	-6
G	-2	-2	5	-2	-6
T	-2	-2	-2	5	-6
$\_$	-6	-6	-6	-6	0

where the value -2 at in row 1, column 2 corresponds to matching the symbol `A` from x to the symbol `C` from y, the sequences

```
AGAT
GATA
```

have the optimal score of 3 and the optimal matching of

```
AGAT_
_GATA.
```

The *score* and *trace* matrixes corresponding to the sequences are

0	-6	-12	-18	-24
-6	-2	-1	-7	-13
-12	-1	-4	-3	-9
-18	-7	4	-2	2
-24	-13	-2	9	**3**

Table 3: *score* matrix for sequences `AGAT` and `GATA`

0	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
$\uparrow$	$\nwarrow$	$\nwarrow$	$\leftarrow$	$\nwarrow$
$\uparrow$	$\nwarrow$	$\nwarrow$	$\nwarrow$	$\nwarrow$
$\uparrow$	$\uparrow$	$\nwarrow$	$\leftarrow$	$\nwarrow$
$\uparrow$	$\uparrow$	$\uparrow$	$\nwarrow$	$\leftarrow$

Table 4: *trace* matrix for sequences `AGAT` and `GATA`

According to the path taken from $(0, 0)$ to $(4, 4)$ using the reverse direction of the arrows in *trace*, the sequence of arrows is `down-diagonal-diagonal-diagonal-right`, which gives us the optimal matching as required.

2.4 Complexity Analysis

Due to the nature of the algorithm used, it is highly similar to the LCS problem. Hence, the complexity of the algorithm remains $O(mn)$, where m and n are the lengths of the given sequences.

2.5 Pseudocode

Algorithm 1: `PairwiseAlignment`

Input: (x, y, s), two sequences, point scheme
Result: $(score, xMod, yMod)$, such that x and y are modified to be optimally aligned

$m \leftarrow$ *length(x)*, $n \leftarrow$ *length(y)* ;
score $\leftarrow$ *arr*$[m+1][n+1]$;
trace $\leftarrow$ *arr*$[m+1][n+1]$;

$score[0][0] \leftarrow 0$;
for *i in 1 to m* **do**
  $score[i][0] \leftarrow score[i-1][0] + \mu(x_i, \_)$;
  $trace[i][0] \leftarrow$ ' $\uparrow$ ' ;
end
for *j in 1 to n* **do**
  $score[0][j] \leftarrow score[0][j-1] + \mu(\_, y_j)$;
  $trace[0][j] \leftarrow$ ' $\leftarrow$ ' ;
end

for *i in 1 to m* **do**
  for *j in 1 to n* **do**
    diagonalValue $\leftarrow score[i-1][j-1] + \mu(x_i, y_j)$;
    leftValue $\leftarrow score[i][j-1] + \mu(\_, y_j)$;
    topValue $\leftarrow score[i-1][j] + \mu(x_i, \_)$;
    if $(diagonalValue > leftValue)$ *and* $(diagonalValue > topValue)$ **then**
      $score[i][j] \leftarrow diagonalValue$;
      $trace[i][j] \leftarrow$ ' $\nwarrow$ ' ;
    end
    else if *leftValue* > *topValue* **then**
      $score[i][j] \leftarrow leftValue$;
      $trace[i][j] \leftarrow$ ' $\leftarrow$ ' ;
    end
    else
      $score[i][j] \leftarrow topValue$;
      $trace[i][j] \leftarrow$ ' $\uparrow$ ' ;
    end
  end
end
$score \leftarrow score[m][n]$;

$i \leftarrow m$;
$j \leftarrow n$;
while $i > 0$ *or* $j > 0$ **do**
  if $trace[i][j] =$ ' $\nwarrow$ ' **then**
    $xMod \leftarrow xMod + x_i$;
    $yMod \leftarrow yMod + y_j$;
    $i \leftarrow i - 1$;
    $j \leftarrow j - 1$;
  end
  else if $trace[i][j] =$ ' $\leftarrow$ ' **then**
    $xMod \leftarrow xMod + \_$;
    $yMod \leftarrow yMod + y_j$;
    $j \leftarrow j - 1$;
  end
  else
    $xMod \leftarrow xMod + x_i$;
    $yMod \leftarrow yMod + \_$;
    $i \leftarrow i - 1$;
  end
end
return $(score, xMod, yMod)$;