Thomas Freeman

2/14/22

Bioinformatics for CS

Professor Kim

Report for HW3

**Abstract:**

The problem stated is to write a program that can analyze pairs of DNA strings and return an alignment score. This program revolves around using dynamic programming techniques to align different strings of DNA, with a score that is optimal to both given strings. The table given in HW3 shows how the strings will be scored and is implemented. There are multiple approaches to alignment of strings as we have discussed in class. Below I will discuss the methodology, concepts, and how my own results can differ from the findings of other methods.

**Method:**

The algorithm I chose to implement was the Needleman-Wunsch or global alignment algorithm. The programming for the main method is in three parts. Needleman involves the initialization of a n x 2 matrix with initial string values, Filling the matrix with values from both strings on different sides of the matrix, then backtracking through the matrix to determine optimal alignment. The matrix is filled in reverse, going from the lower right most character to the upper left.

For the scoring system, this algorithm has gap penalty or opening implemented, which helps with scoring how aligned the sequences are. Showing gaps in the sequence allows the global alignment to match more terms. I have also implemented penalties for extension of a string and mismatches, extension is treated like a gap opening, so they have the same value. a match is worth 20 points, mismatches are worth -10, and I decided that gaps and extensions would be worth -5 points. The data for the algorithm is read from a separate text file called test, containing the two strings on top of each other.

**Discussion:**

When we set mismatches to 0 and just consider the extensions and openings, A higher g will generally decrease the score of the alignment. For example, with the first set of strings, There’s very little in terms of characters to be analyzed. The values for gaps and extensions can change these arbitrary scores drastically. We can adjust these values such that were only looking for what we want to see. We could just want to look for matches or look for all kinds of mismatches in the strings. Using values like –20 and –50 for g gave much lower scores then what’s seen in the results. However, what if we want to define what the perfect alignment score is, as all these values are completely arbitrary.

If we have a lot of matches and want to have the lowest score possible, then we could increase values for gap and extension if necessary. We can use different scorings to validate the similarity of the strings depending on their levels of complexity. For strings of say 50 plus characters could be subject to a different scoring system then those with less then 10 characters. The value of perfect alignment changes with the scores. With the system that I have, perfect alignment is equal to the length of the string times 20 since that implies all character’s match. In another case there could be exaggerated scores for negative attributes like gap opening and mismatches, such that the perfect score would be getting as close to 0 as possible. This all depends on what the developer values more in the optimal alignment.

**Results:**

The following figures show the output for each set of test strings. For the figure 1 we have 5 distinct matches marked by the vertical lines between matching amino acids. Since we have 5 of those, we get a total match score of 100. We then have two mismatches and one gap extension with a total of -25 so we get a total score of 75. The same calculation applies to figure 2 where we have 14 matches to get a match score of 280, minus 11 mismatches with one gap extension to get -115, ending with a total score of 165.

**Figure 1 Figure 2**

Text

Description automatically generatedText

Description automatically generated with medium confidence

**Appendix:**

The following is the code for the assignment

# Thomas Freeman 2/11/22

# implementation of the Needleman-Wunsch algorithm

# to count gaps in strings.

import logging

import argparse

def parse(fname):

with open(fname) as fp:

lines = fp.readlines()

return lines[0].rstrip(), lines[1]

def get\_args():

parser = argparse.ArgumentParser(description='Global sequence alignment')

parser.add\_argument('-i', '--input', dest='filename', type=str, help='Input filename', default="./test.txt")

parser.add\_argument('-a', '--algorithm', dest='algorithm', type=str, default="global")

parser.add\_argument('-m', '--match', dest='match\_score', type=int, help='Match score', default=20)

parser.add\_argument('-p', '--miss', dest='miss\_penalty', type=int, help='Miss match penalty', default=-10)

parser.add\_argument('-o', '--gap\_opening', dest='gap\_opening', type=int, help='Gap opening penalty', default=-5)

parser.add\_argument('-e', '--gap\_extension', dest='gap\_extension', type=int, help='Gap extension penalty', default=-5)

args = parser.parse\_args()

return args

def init\_matrix(rows, columns, algorithm='global'):

mat = [[0] \* columns for row in range(rows)]

gap\_mat = [[[0, set()]] \* columns for row in range(rows)]

gap\_mat[1][0] = [0, set()]

gap\_mat[0][1] = [0, set()]

# fills the cells in the first row and first column

for i in range(1, rows):

mat[i][0] = mat[i - 1][0] + gap\_penalty(gap\_mat, i - 1, 0, 'v')

gap\_mat[i][0] = [1, {'v'}]

for j in range(1, columns):

mat[0][j] = mat[0][j - 1] + gap\_penalty(gap\_mat, 0, j - 1, 'h')

gap\_mat[0][j] = [1, {'h'}]

return mat, gap\_mat

def print\_matrix(matrix):

for row in matrix:

print(''.join(['{0:>{w}}'.format(item, w=5) for item in row]), end='\n\n')

def is\_match(char\_a, char\_b):

return match\_score if char\_a == char\_b else miss\_match\_penalty

def gap\_penalty(gap\_matrix, row\_index, col\_index, gap\_direction):

# If there's a gap detected, we determine if

# this gap is between both strings or

# between characters of one string.

if gap\_matrix[row\_index][col\_index][0] == 1:

if gap\_direction == 'v':

if 'v' in gap\_matrix[row\_index][col\_index][1]:

return gap\_extend

elif gap\_direction == 'h':

if 'h' in gap\_matrix[row\_index][col\_index][1]:

return gap\_extend

return gap\_opening

# Method for the Needleman-Wunsch Algorithm

def global\_alignment(M, gap\_matrix, a, b, rows, columns):

# fills the score matrix

for i in range(1, rows):

for j in range(1, columns):

diagonal = M[i - 1][j - 1] + is\_match(a[j - 1], b[i - 1])

vgap = M[i - 1][j] + gap\_penalty(gap\_matrix, i - 1, j, 'v')

hgap = M[i][j - 1] + gap\_penalty(gap\_matrix, i, j - 1, 'h')

options = [diagonal, vgap, hgap]

index\_max = options.index(max(options))

if options[index\_max] == hgap:

gap\_matrix[i][j] = [1, {'h'}]

if options[index\_max] == vgap:

gap\_matrix[i][j] = [1, {'v'}]

M[i][j] = options[index\_max]

i, j = rows - 1, columns - 1

aligned\_a, aligned\_b, mid = (' ') \* 3

# backtrack process from lower right to upper left of the matrix

while i > 0 and j > 0:

diagonal = M[i][j] - is\_match(a[j - 1], b[i - 1])

vgap = M[i][j] - gap\_penalty(gap\_matrix, i - 1, j, 'v')

hgap = M[i][j] - gap\_penalty(gap\_matrix, i, j - 1, 'h')

if M[i - 1][j - 1] == diagonal:

aligned\_a += a[j - 1]

aligned\_b += b[i - 1]

if is\_match(a[j - 1], b[i - 1]) == match\_score:

mid += '|'

else:

mid += ' '

i = i - 1

j = j - 1

elif M[i - 1][j] == vgap:

aligned\_a += '-'

aligned\_b += b[i - 1]

mid += ' '

i = i - 1

elif M[i][j - 1] == hgap:

aligned\_a += a[j - 1]

aligned\_b += '-'

mid += ' '

j = j - 1

while j > 0:

aligned\_a += a[j - 1]

aligned\_b += '-'

mid += ' '

j = j - 1

while i > 0:

aligned\_a += '-'

aligned\_b += b[i - 1]

mid += ' '

i = i - 1

# The Aligned arrays are printed.

print(aligned\_a[::-1] + '\n' + mid[::-1] + '\n' + aligned\_b[::-1], "\n")

match\_hit = mid.count('|')

return M, M[rows - 1][columns - 1]

if \_\_name\_\_ == '\_\_main\_\_':

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

args = get\_args()

if args.filename:

filename = args.filename

sequence\_a, sequence\_b = parse(filename)

rows, columns = len(sequence\_b) + 1, len(sequence\_a) + 1

match\_score = args.match\_score

miss\_match\_penalty = args.miss\_penalty

gap\_opening = args.gap\_opening

gap\_extend = args.gap\_extension

print("Filename\t:{}\nAlgorithm\t:{}\nMatch score\t:{}\nMiss match\t:{}\nGap opening\t:{}\nGap Extension\t:{}\n".format

(args.filename, args.algorithm, match\_score, miss\_match\_penalty, gap\_opening, gap\_extend))

if args.algorithm == 'global':

D, gap\_matrix = init\_matrix(rows, columns, algorithm=args.algorithm)

D, score = global\_alignment(D, gap\_matrix, sequence\_a, sequence\_b, rows, columns)

print("Total Alignment Score:", score)