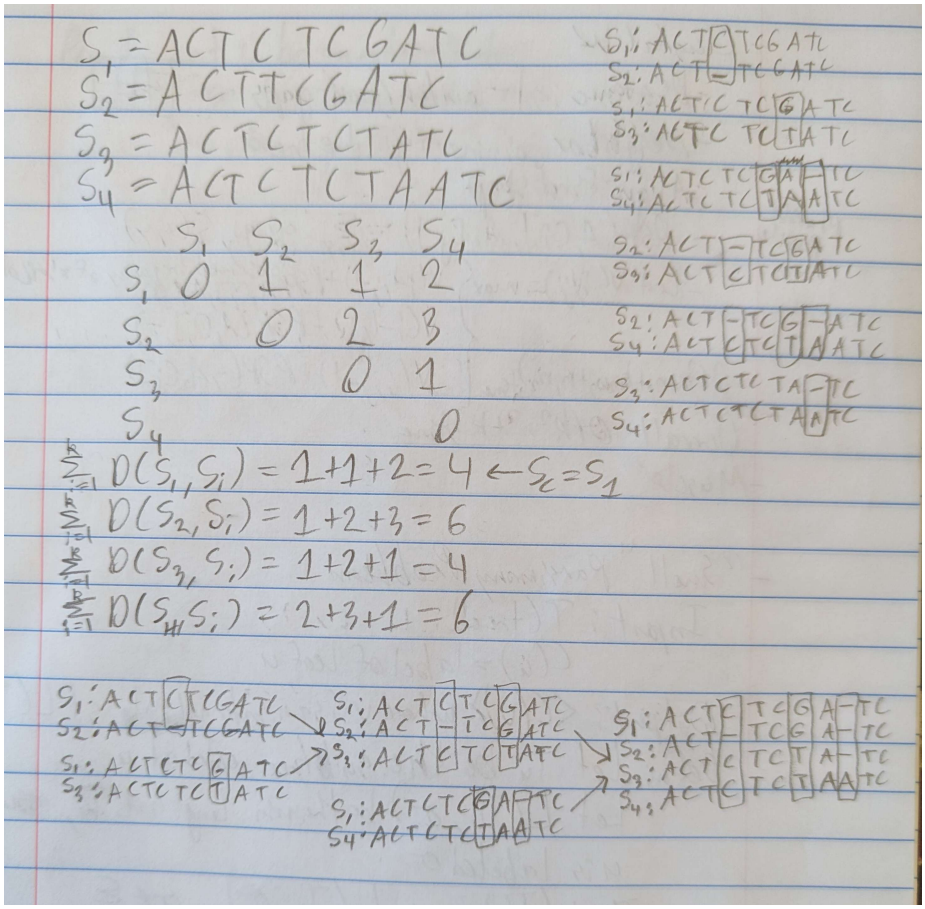
HW 5

Kelly Joyce, Justin Mau, Benjamin Haedt

November 2023

1 Problem 1

Explanation on the next page.



The first step is to create the matrix. To do this, each pair of sequences was compared (through coding it would be done through global sequence alignment). Whenever there was an insertion/deletion or a mismatch, a point would be added to the score of that pair. It could then be plotted into the matrix. Only half of the matrix needs to be filled as $S_1 to S_2$ is the same score as $S_2 to S_1$. There is a diagonal of 0s as every sequence compared with itself needs no modifications, therefore would have a value of zero.

With this, the distance was found of each sequence. This was calculated by taking the sum of all of the numbers that are present in the matrix for said sequence, including both the row and column of the sequence. The center was then assigned by picking the one with the smallest distance. In this case, both S_1 and S_3 have the smallest distance of 4, so S_1 was picked out of the two.

After this, the global sequence alignments of the center (S_1) with the other sequences were used to complete the final alignment. Noting where the differences are through boxes, each of the alignments with the center were aligned together one at a time. If there is a mismatch, there will not be much adjusting besides noting it exists. However, insertions or deletions may expand the overall sequence alignment. This is the case when adding S_4 to the overall alignment. For this, an insertion was added to every other sequence to fit the alignment.

2 Problem 2

General Setting Parameters:

Output Format: CLUSTAL

Pairwise Alignment: **FAST/APPROXIMATE** **SLOW/ACCURATE**

Enter your sequences (with labels) below (copy & paste): **PROTEIN** **DNA**

Support Formats: FASTA (Pearson), NBRF/PIR, EMBL/Swiss Prot, GDE, CLUSTAL, and GCG/MSF

```
>s1
ACTCTCGATC
>s2
ACTTCGATC
>s3
ACTCTCTATC
>s4
ACTCTCTAATC
```

Or give the file name containing your query

Choose File No file chosen

Execute Multiple Alignment Reset

More Detail Parameters...

Pairwise Alignment Parameters:

For FAST/APPROXIMATE:

K-tuple(word) size: 2, Window size: 4, Gap Penalty: 5

Number of Top Diagonals: 5, Scoring Method: PERCENT

For SLOW/ACCURATE:

Gap Open Penalty: 15, Gap Extension Penalty: 6.66

Select Weight Matrix: IUB (for DNA)

(Note that only parameters for the algorithm specified by the above "Pairwise Alignment" are valid.)

Multiple Alignment Parameters:

Gap Open Penalty: 15, Gap Extension Penalty: 6.66

Weight Transition: **YES (Value:** 0.5 **),** **NO**

Hydrophilic Residues for Proteins:

Hydrophilic Gaps: **YES** **NO**

Select Weight Matrix: IUB (for DNA)

Type additional options (delimited by whitespaces) below:

(**-options** for help)

Execute Multiple Alignment Reset

Feedback **KEGG** **GenomeNet** Kyoto University Bioinformatics Center

```
 CLUSTAL 2.1 Multiple Sequence Alignments


Sequence type explicitly set to DNA
Sequence format is Pearson
Sequence 1: s1            10 bp
Sequence 2: s2             9 bp
Sequence 3: s3            10 bp
Sequence 4: s4            11 bp
Start of Pairwise alignments
Aligning...



Sequences (1:2) Aligned. Score: 66.6667
Sequences (1:3) Aligned. Score: 90
Sequences (1:4) Aligned. Score: 60
Sequences (2:3) Aligned. Score: 55.5556
Sequences (2:4) Aligned. Score: 55.5556
Sequences (3:4) Aligned. Score: 80
Guide tree file created:   [clustalw.dnd]

There are 3 groups
Start of Multiple Alignment

Aligning...
Group 1: Sequences:   2      Score:161
Group 2: Sequences:   3      Score:149
Group 3: Sequences:   4      Score:102
Alignment Score 307

CLUSTAL-Alignment file created  [clustalw.aln]
```

clustalw.aln

```
CLUSTAL 2.1 multiple sequence alignment


s3                ACTCTCTATC-
s4                ACTCTCTAATC
s1                ACTCTCGATC-
s2                ACT-TCGATC-
                  *** ** *
```

clustalw.dnd

```
(
(
s1:0.06944,
s2:0.26389)
:0.08056,
s3:0.02500,
s4:0.17500);
```

Select tree menu Exec

ClustalW server used: https://www.genome.jp/tools-bin/clustalw

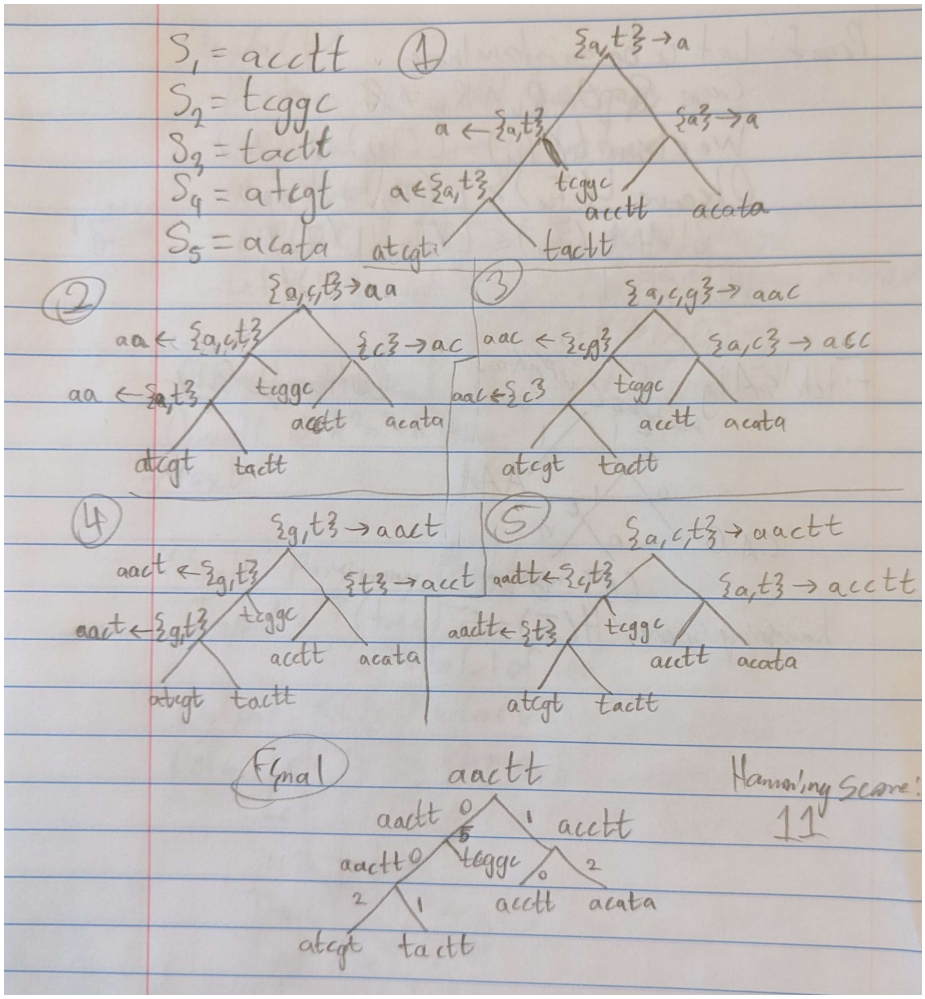
3 Problem 3

To accomplish this, local alignment is performed in k dimensions where k is the number of sequences provided. This means that Smith-Waterman was used to create a k-dimension matrix. All of the sides are given a value of zero, and the point value of each slot in the matrix is the max of the following factors if given 2 sequences S and T.
$V[i-1, j-1] + \delta(S[i], T[j])$
$V[i-1, j] + \delta(S[i],_)$
$V[i, j-1] + \delta(_, T[j])$ or
0

However, as this algorithm is applying for more than two sequences, these factors can expand to include all combinations of match/mismatches, insertions, and deletions between k sequences. This results in a matrix in k dimensions. To find the optimal local alignment, the position with the largest value was found, and then worked backwards until the value of the position in said matrix is 0. Depending on which path this is, the insertions/deletions, matches, and mismatches can be written.

The big-O for this is $O(n^k)$ where n represents the length of the longest sequence and k represents the number of sequences given.

4 Problem 4


S1 = acctt
S2 = tcggc
S3 = tactt
S4 = atcgt
S5 = acata
Final
aactt
Hamming Score:
11

5 Problem 5 Screenshots

```
No command line args... using default values for a, b, c
Reading in FASTA file...
S1: ACTCTCGATC
S2: ACTTCGATC
S3: ACTCTCTATC
S4: ACTCTCTAATC


    | S1 | S2 | S3 | S4 |
_____________________________
 S1 | 0.0  | -1.0  | -1.0  | -2.0  |
_____________________________
 S2 | -1.0  | 0.0  | -2.0  | -3.0  |
_____________________________
 S3 | -1.0  | -2.0  | 0.0  | -1.0  |
_____________________________
 S4 | -2.0  | -3.0  | -1.0  | 0.0  |

[4.0, 6.0, 4.0, 6.0]
Found a center sequence: Sequence(name='s1', sequence='ACTCTCGATC')

On alignment with s4, insertion(s) added to center sequence...

Multiple Sequence Alignment -------------
s1: ACTCTC_GATC
s2: ACT_TC_GATC
s3: ACTCTC_TATC
s4: ACTCTCTAATC
```

```
No command line args... using default values for a, b, c
Reading in FASTA file...
S1: CCTGCTGCAG
S2: GATGTGCCG
S3: GATGTGCAG
S4: CCGCTAGCAG
S5: CCTGTAGG


    | S1 | S2 | S3 | S4 | S5 |
_____________________________
 S1 | 0.0  | -4.0  | -3.0  | -2.0  | -4.0  |
_____________________________
 S2 | -4.0  | 0.0  | -1.0  | -6.0  | -5.0  |
_____________________________
 S3 | -3.0  | -1.0  | 0.0  | -5.0  | -5.0  |
_____________________________
 S4 | -2.0  | -6.0  | -5.0  | 0.0  | -4.0  |
_____________________________
 S5 | -4.0  | -5.0  | -5.0  | -4.0  | 0.0  |

[13.0, 16.0, 14.0, 17.0, 18.0]
Found a center sequence: Sequence(name='s1', sequence='CCTGCTGCAG')

On alignment with s4, insertion(s) added to center sequence...

Multiple Sequence Alignment ------------
s1: CCTGCT_GCAG
s2: GATG_T_GCCG
s3: GATG_T_GCAG
s4: CC_GCTAGCAG
s5: CCTG_TAG__G
PS C:\Users\Jdmau\OneDrive\Documents\Documents\Fall 2023\CSCI 551 - Adv Comp Bio\HW5>
```

```
No command line args... using default values for a, b, c
Reading in FASTA file...
S1: AAAGGGCCCTTT
S2: AGCTAGCT
S3: AACGTACGTAT
S4: TACTACTACT


    | S1 | S2 | S3 | S4 |
_____________________________
 S1 | 0.0  | -8.0  | -6.0  | -9.0  |
_____________________________
 S2 | -8.0  | 0.0  | -5.0  | -5.0  |
_____________________________
 S3 | -6.0  | -5.0  | 0.0  | -4.0  |
_____________________________
 S4 | -9.0  | -5.0  | -4.0  | 0.0  |

[23.0, 18.0, 15.0, 18.0]
Found a center sequence: Sequence(name='s3', sequence='AACGTACGTAT')

On alignment with s4, insertion(s) added to center sequence...

Multiple Sequence Alignment -------------
s1: AAGGGCCCTT_T
s2: AGC_TA_G_C_T
s3: AACGTACGTA_T
s4: TAC_TAC_TACT
```

6 Problem 5 Code

```
import numpy as np
class Needleman_Wunsch:
    def __init__(self, seq1, seq2, a, b, c) -> None:
        self.S, self.T        = seq1, seq2
        self.a, self.b, self.c = a, b, c
        self.n, self.m        = len(self.S), len(self.T)
        self.alignmentScore = None
        self.matrix = np.zeros((self.m + 1, self.n + 1))
        self.path = np.zeros((self.m + 1, self.n + 1))

    def getAlignmentScore(self):
        return self.alignmentScore

    def findAlignment(self):
        self.initializeBoundaries()
        self.run()
        return self.alignmentScore


    # initialize boundaries
```

```
def initializeBoundaries(self):
    for i in range(1, self.m + 1):
        self.matrix[i][0] = self.matrix[i-1][0] + self.c
        self.path[i][0] = 1 # from above

    for j in range(1, self.n + 1):
        self.matrix[0][j] = self.matrix[0][j-1] + self.c
        self.path[0][j] = 2 # from left


def run(self):
    for i in range(1, self.m + 1):
        for j in range(1, self.n + 1):

            # find max of three options
            options = []
            if(self.T[i-1] == self.S[j-1]):
                options.append(self.matrix[i-1, j-1] + self.a) # match
            else:
                options.append(self.matrix[i-1, j-1] + self.b) #
                    mismatch
            options.append(self.matrix[i-1, j] + self.c) #
                insertion/deletion
            options.append(self.matrix[i, j-1] + self.c) #
                insertion/deletion

            self.matrix[i, j] = max(options)

            # 0 for diagonal
            # 1 for above,
            # 2 for left
            self.path[i, j] = np.argmax(options)

    # print(self.alignmentScore)
    self.alignmentScore = self.matrix[self.m, self.n]


def rebuildAlignment(self, isMSA=False, verbose=False) :

    i = self.m
    j = self.n

    S_align = ""
    T_align = ""

    # for MSA, T will be the centerSequence

    if verbose:
        print("\nRebuilding...")
        print(f"S: |{self.S}| = {self.n}")
```

```
            print(f"T: |{self.T}| = {self.m}")

        while(i > 0 and j > 0):
            if (self.path[i][j] == 0):   # 0 for diagonal (match/mismatch)
                S_align += self.S[j - 1]
                T_align += self.T[i - 1]
                i -= 1
                j -= 1

            elif (self.path[i][j] == 1): # 1 for above, (insertion _ in S)
                S_align += "_"
                T_align += self.T[i - 1]
                i -= 1
                # j

            # centerSequence insertion, need to take note and add to all
                 previous in MSA
            elif (self.path[i][j] == 2): # 2 for left  (insertion _ in T)
                S_align += self.S[j - 1]
                T_align += "_"
                # i
                j -= 1

        if verbose:
            print(S_align[::-1])
            print(T_align[::-1])

        S_align = S_align[::-1]
        T_align = T_align[::-1]

        return (S_align, T_align)


from typing import NamedTuple

class Sequence(NamedTuple):
    name: str
    sequence: str

# read in FASTA file (will read in any number of sequences)
def readFASTA(filename):
    sequenceList = []
    print("Reading in FASTA file...")
    f = open(filename, 'r')
    lines = f.readlines()

    readingSequence = False
    name = ""
    seq = ""
    for line in lines:
```

```
        line = line.strip('\n')
        if (line[0] == '>'):
            if (readingSequence == True):
                sequenceList.append(Sequence(name, seq)) # finishes the
                    current sequence
                seq = ""   # resets the sequence to be blank
            else:
                readingSequence = True

            name = line.strip('> ')

        else:
            seq += line

    # after the end of file is reached, it will add that last sequence
    sequenceList.append(Sequence(name, seq))

    f.close()
    return sequenceList


from Needleman_Wunsch import Needleman_Wunsch
from Sequence import Sequence, readFASTA
import sys

sequenceList = []

def printAlignmentGrid(grid, sequenceList):
    print("\n   |", end="")
    for i, s in enumerate(sequenceList):
        print(f" S{i+1} |", end="")
    print()

    for i, s in enumerate(grid):
        print("------------------------------")
        print(f" S{i+1} |", end="")
        for j, t in enumerate(grid[i]):
            print(f" {t} |", end="")
        print()


def printMSA(MSA):
    print(f"\nMultiple Sequence Alignment -------------")
    for i, s in enumerate(MSA):
        print(f"s{i+1}: {s}")

# start of main() ----------------------------\
def main():
    print()
    if (len(sys.argv) == 5):
```

```
    a = int(sys.argv[1])
    b = int(sys.argv[2])
    c = int(sys.argv[3])
    filename = sys.argv[4]

else:
    print("No command line args... using default values for a, b, c")
    a, b, c = 0, -1, -1
    filename = "default.fasta"

sequenceList = readFASTA(filename)

for i, s in enumerate(sequenceList):
    print(f"S{i+1}: {s.sequence}")

alignmentsGrid = []
for i, s in enumerate(sequenceList):
    alignmentsGrid.append([])
    for s in sequenceList:
        alignmentsGrid[i].append(0)

sumOfScores = []

for i, s in enumerate(sequenceList):
    for j, t in enumerate(sequenceList):
        alignment = Needleman_Wunsch(s.sequence, t.sequence, a,b,c)
        alignmentsGrid[i][j] = alignment.findAlignment()
    sumOfScores.append(sum(alignmentsGrid[i]) * -1)

print()
printAlignmentGrid(alignmentsGrid, sequenceList)
print(f"\n{sumOfScores}")

centerIndex = sumOfScores.index(min(sumOfScores))
centerSequence = sequenceList[centerIndex]
print(f"Found a center sequence: {centerSequence}\n")

MSA = []
for s in enumerate(sequenceList):
    MSA.append("")
MSA[centerIndex] = centerSequence.sequence

for i, s in enumerate(sequenceList):
    if s != centerSequence:
        alignment = Needleman_Wunsch(s.sequence, MSA[centerIndex],
            a,b,c)
        alignment.findAlignment()
        alignment = alignment.rebuildAlignment(isMSA=True)
        MSA[i] = alignment[0]
        # check if insertions made to centerSequence
```

```
            if (MSA[centerIndex] != alignment[1]):
                print(f"On alignment with s{i+1}, insertion(s) added to
                    center sequence...")
                for j, char in enumerate(alignment[1]):
                    if(char == "_"):
                        for k in range(0, i):
                            if(k != centerIndex):
                                MSA[k] = MSA[k][:j] + "_" + MSA[k][j:]

                MSA[centerIndex] = alignment[1]

    printMSA(MSA)

# end of main() ------------------------------\


if __name__ == '__main__':
    main()
```