1 BIO213 - Introduction to Quantitative Biology

1.0.1 Assignment 1

Name: Ansh Arora **Roll No:** 2019022

QUESTION 1

Find the best global alignment between DNA sequences ATCAGAGTA and TTCAGTA using scoring scheme - - Match = +2 - Mismatch = -1 - Gap = -1

[94]:
```
def matprint(dp):
    for row in dp:
        for x in row:
            if x<0:
                print(x, end=" ")
            else:
                print(" "+str(x), end = " ")
        print()
```

ANSWER 1

We will first be initiating the sequences and their lengths correctly.

[115]:
```
s1 = "ATCAGAGTA"
s2 = "TTCAGTA"

l1 = len(s1)
l2 = len(s2)

print("s1: "+s1+"\ns1 length: "+str(l1))
print("s2: "+s2+"\ns2 length: "+str(l2))
```

```
s1: ATCAGAGTA
s1 length: 9
s2: TTCAGTA
s2 length: 7
```

[116]:
```
match = 2
mismatch = -1
gap = -1
```

Now that we have assigned our scoring values to variables, we will now create our DP matrix and initialize it with values for the 0th row and 0th column.

[117]:
```
dp = [[0 for i in range(l2+1)] for j in range(l1+1)]
#matprint(dp)
```

[118]:
```
for i in range(1,l2+1):
    dp[0][i] = dp[0][i-1]+gap
```

```
for i in range (1,l1+1):
    dp[i][0] = dp[i-1][0]+gap

#matprint(dp)
```

ANSWER A) Now we will fill up our matrix using Dynamic Programming with the rules taught to us in the class.

[119]:
```
for i in range(1,l1+1):
    for j in range(1,l2+1):
        if(s1[i-1]==s2[j-1]):
            #Match case
            op1 = match + dp[i-1][j-1]
            op2 = gap + dp[i-1][j]
            op3 = gap + dp[i][j-1]
            dp[i][j] = max(op1,op2,op3)
        else:
            op1 = mismatch + dp[i-1][j-1]
            op2 = gap + dp[i-1][j]
            op3 = gap + dp[i][j-1]
            dp[i][j] = max(op1,op2,op3)

print("The DP Matrix will be- ")
matprint(dp)
```

```
The DP Matrix will be-
 0 -1 -2 -3 -4 -5 -6 -7
-1 -1 -2 -3 -1 -2 -3 -4
-2  1  1  0 -1 -2  0 -1
-3  0  0  3  2  1  0 -1
-4 -1 -1  2  5  4  3  2
-5 -2 -2  1  4  7  6  5
-6 -3 -3  0  3  6  6  8
-7 -4 -4 -1  2  5  5  7
-8 -5 -2 -2  1  4  7  6
-9 -6 -3 -3  0  3  6  9
```

Now we have to recreate the alignment from the DP matrix. This can be done using backtracking on the matrix, starting from the bottom right cell and progression optimally till we reach the top right cell.

ANSWER B) Yes there is a possibility of that there are more than one ways of aligining the given sequences, since at places where the score might be coming from a match from the cell located diagonally to it, it can also come from a gap resulting from a horizontal or vertical cell if the gap penalty added back gives that value. Similarly, we can do the same for a mismatch and gaps. We will find all these sequences in the following backtracking code, which uses recursion. All cases will thus be considered here.

ANSWER C)

[120]:
```
def backtrack(dp,row,col,ans1,ans2):
    #print(str(row)+" "+str(col))
    if row==0 and col==0:
        #print(str(row)+" "+str(col))
        print("Score: " +str(dp[l1][l2]))
        print(ans1)
        for i in range(len(ans1)):
            if ans1[i]==ans2[i]:
                print("|",end="")
            else:
                print(" ",end="")
        print()
        print(ans2)
        print()
        return
    if(s1[row-1]!=s2[col-1]):
        if dp[row][col] == dp[row-1][col-1]+mismatch:
            backtrack(dp,row-1,col-1,s1[row-1]+ans1,s2[col-1]+ans2)
        if dp[row][col] == dp[row-1][col]+gap:
            backtrack(dp,row-1,col,s1[row-1]+ans1,"-"+ans2)
        if dp[row][col] == dp[row][col-1]+gap:
            backtrack(dp,row,col-1,"-"+ans1,s2[col-1]+ans2)

    else:
        backtrack(dp,row-1,col-1,s1[row-1]+ans1,s2[col-1]+ans2)
        if dp[row][col] == dp[row-1][col]+gap:
            backtrack(dp,row-1,col,s1[row-1]+ans1,"-"+ans2)
        if dp[row][col] == dp[row][col-1]+gap:
            backtrack(dp,row,col-1,"-"+ans1,s2[col-1]+ans2)

print("Optimal alingments - ")
print()
backtrack(dp,l1,l2,"","")
```

```
Optimal alingments -

Score: 9
ATCAGAGTA
 ||  ||||
TTC--AGTA

Score: 9
ATCAGAGTA
 |||  |||
TTCA--GTA
```

```
Score: 9
ATCAGAGTA
 ||||  ||
TTCAG--TA
```

QUESTION 2

Using the above sequences and scoring scheme, compute the most optimal local alignment.

ANSWER 2

We will first formulate a DP matrix for local alignment, where no values smaller than 0 can appear.

[121]:
```
dp2 = [[0 for i in range(l2+1)] for j in range(l1+1)]
#matprint(dp2)
```

ANSWER A) We will now fill the DP Matrix up with new for loops keeping the 0 property in mind

[122]:
```
for i in range(1,l1+1):
    for j in range(1,l2+1):
        if(s1[i-1]==s2[j-1]):
            #Match case
            op1 = match + dp2[i-1][j-1]
            op2 = gap + dp2[i-1][j]
            op3 = gap + dp2[i][j-1]
            dp2[i][j] = max(op1,op2,op3,0)
        else:
            op1 = mismatch + dp2[i-1][j-1]
            op2 = gap + dp2[i-1][j]
            op3 = gap + dp2[i][j-1]
            dp2[i][j] = max(op1,op2,op3,0)

print("The DP Matrix will be- ")
matprint(dp2)
```

```
The DP Matrix will be-
 0  0  0  0  0  0  0  0
 0  0  0  0  2  1  0  2
 0  2  2  1  1  1  3  2
 0  1  1  4  3  2  2  2
 0  0  0  3  6  5  4  4
 0  0  0  2  5  8  7  6
 0  0  0  1  4  7  7  9
 0  0  0  0  3  6  6  8
 0  2  2  1  2  5  8  7
 0  1  1  1  3  4  7  10
```

We now need to find the max element locations in the matrix to start backtracking and store them in maxloc array for local alignment. For this we will run the following loop -

[124]:
```
maxloc = []
maxval = 0

for i in range(l1+1):
    for j in range(l2+1):
        if dp2[i][j]==maxval:
            maxloc.append((i,j))
        elif dp2[i][j]>maxval:
            maxloc.clear()
            maxloc.append((i,j))
            maxval = dp2[i][j]

#print(maxloc)
#print(maxval)
```

ANSWER B) Now we will run backtracking from the achieved maxrow and maxcol values until we hit a value of 0 in the DP Table, thus achieving our optimal local alignment.

[125]:
```
def backtrack(dp,row,col,ans1,ans2):
    #print(str(row)+" "+str(col)+" "+str(dp[row][col]))
    if dp[row][col]==0:
        #print(str(row)+" "+str(col))
        print("Score: " +str(dp[maxrow][maxcol]))
        print(ans1)
        for i in range(len(ans1)):
            if ans1[i]==ans2[i]:
                print("|",end="")
            else:
                print(" ",end="")
        print()
        print(ans2)
        print()
        return
    if(s1[row-1]!=s2[col-1]):
        if dp[row][col] == dp[row-1][col-1]+mismatch:
            backtrack(dp,row-1,col-1,s1[row-1]+ans1,s2[col-1]+ans2)
        if dp[row][col] == dp[row-1][col]+gap:
            backtrack(dp,row-1,col,s1[row-1]+ans1,"-"+ans2)
        if dp[row][col] == dp[row][col-1]+gap:
            backtrack(dp,row,col-1,"-"+ans1,s2[col-1]+ans2)

    else:
        backtrack(dp,row-1,col-1,s1[row-1]+ans1,s2[col-1]+ans2)
        if dp[row][col] == dp[row-1][col]+gap:
            backtrack(dp,row-1,col,s1[row-1]+ans1,"-"+ans2)
        if dp[row][col] == dp[row][col-1]+gap:
            backtrack(dp,row,col-1,"-"+ans1,s2[col-1]+ans2)
```

```
print("Optimal alingments - ")
print()
for i in maxloc:
    backtrack(dp2,i[0],i[1],"","")
```

```
Optimal alingments -

Score: 10
TCAGAGTA
||  ||||
TC--AGTA

Score: 10
TCAGAGTA
|||  |||
TCA--GTA

Score: 10
TCAGAGTA
||||  ||
TCAG--TA
```

QUESTION 3

Changes that were required in the program in order to perform local rather than global pairwise sequence alignment.

ANSWER 3

To perform local pairwise sequence alignment, the following changes were made to the global alignment code - 1. After initiating a DP matrix for local alignment, the first row and first column were left as 0, while in the case of global they were changed to accomodate for gap penalties. 2. While the three option cases we take for filling each cell remained the same for both, in local the max value to be filled in the cell was taken to be max(0,op1,op2,op3) rather than just max(op1,op2,op3) to accomodate for the fact that the minimum value that a cell in the local alignment dp matrix can take is 0. 3. After filling of the matrix, in local alignment we had to find the locations of max score and make a list, whereas in global alignment we simply started from the bottom right corner cell. Thus the backtrack initiation points had to be calculated in case of local alignment. 4. While backtracking, in case of local alignment we stop when we reach a 0, thus identifying a local high alignment case. This is different from the global alignment case where we stop when we reach the top left cell of the matrix.

QUESTION 4

Will changing the scoring scheme to Match = +2, Mismatch = -1, Gap = -2 modify the results obtained in the above questions? If yes, comment on the same.

ANSWER 4

Yes, on changing the scoring scheme to the new one, results including scores and alignments might change to accomodate for the increased gap penalties. We will show this now by applying it on the above sequences -

[126]:
```
match = 2
mismatch = -1
gap = -2
```

Global alignment -

[127]:
```
dp = [[0 for i in range(l2+1)] for j in range(l1+1)]

for i in range(1,l2+1):
    dp[0][i] = dp[0][i-1]+gap

for i in range (1,l1+1):
    dp[i][0] = dp[i-1][0]+gap

for i in range(1,l1+1):
    for j in range(1,l2+1):
        if(s1[i-1]==s2[j-1]):
            #Match case
            op1 = match + dp[i-1][j-1]
            op2 = gap + dp[i-1][j]
            op3 = gap + dp[i][j-1]
            dp[i][j] = max(op1,op2,op3)
        else:
            op1 = mismatch + dp[i-1][j-1]
            op2 = gap + dp[i-1][j]
            op3 = gap + dp[i][j-1]
            dp[i][j] = max(op1,op2,op3)

print("The DP Matrix will be- ")
matprint(dp)
print()

def backtrack2(dp,row,col,ans1,ans2):
    #print(str(row)+" "+str(col))
    if row==0 and col==0:
        #print(str(row)+" "+str(col))
        print("Score: " +str(dp[l1][l2]))
        print(ans1)
        for i in range(len(ans1)):
            if ans1[i]==ans2[i]:
                print("|",end="")
            else:
                print(" ",end="")
        print()
```

```
        print(ans2)
        print()
        return
    if(s1[row-1]!=s2[col-1]):
        if dp[row][col] == dp[row-1][col-1]+mismatch:
            backtrack2(dp,row-1,col-1,s1[row-1]+ans1,s2[col-1]+ans2)
        if dp[row][col] == dp[row-1][col]+gap:
            backtrack2(dp,row-1,col,s1[row-1]+ans1,"-"+ans2)
        if dp[row][col] == dp[row][col-1]+gap:
            backtrack2(dp,row,col-1,"-"+ans1,s2[col-1]+ans2)

    else:
        backtrack2(dp,row-1,col-1,s1[row-1]+ans1,s2[col-1]+ans2)
        if dp[row][col] == dp[row-1][col]+gap:
            backtrack2(dp,row-1,col,s1[row-1]+ans1,"-"+ans2)
        if dp[row][col] == dp[row][col-1]+gap:
            backtrack2(dp,row,col-1,"-"+ans1,s2[col-1]+ans2)

print("Optimal alingments - ")
print()
backtrack2(dp,l1,l2,"","")
```

```
The DP Matrix will be-
 0 -2 -4 -6 -8 -10 -12 -14
-2 -1 -3 -5 -4 -6 -8 -10
-4  0  1 -1 -3 -5 -4 -6
-6 -2 -1  3  1 -1 -3 -5
-8 -4 -3  1  5  3  1 -1
-10 -6 -5 -1  3  7  5  3
-12 -8 -7 -3  1  5  6  7
-14 -10 -9 -5 -1  3  4  5
-16 -12 -8 -7 -3  1  5  3
-18 -14 -10 -9 -5 -1  3  7

Optimal alingments -

Score: 7
ATCAGAGTA
 ||  ||||
TTC--AGTA

Score: 7
ATCAGAGTA
 |||  |||
TTCA--GTA

Score: 7
```

```
ATCAGAGTA
 ||||  ||
TTCAG--TA
```

Local Alignment -

[128]:
```
dp2 = [[0 for i in range(l2+1)] for j in range(l1+1)]
for i in range(1,l1+1):
    for j in range(1,l2+1):
        if(s1[i-1]==s2[j-1]):
            #Match case
            op1 = match + dp2[i-1][j-1]
            op2 = gap + dp2[i-1][j]
            op3 = gap + dp2[i][j-1]
            dp2[i][j] = max(op1,op2,op3,0)
        else:
            op1 = mismatch + dp2[i-1][j-1]
            op2 = gap + dp2[i-1][j]
            op3 = gap + dp2[i][j-1]
            dp2[i][j] = max(op1,op2,op3,0)

print("The DP Matrix will be- ")
matprint(dp2)
print()

maxloc = []
maxval = 0

for i in range(l1+1):
    for j in range(l2+1):
        if dp2[i][j]==maxval:
            maxloc.append((i,j))
        elif dp2[i][j]>maxval:
            maxloc.clear()
            maxloc.append((i,j))
            maxval = dp2[i][j]

#print(maxloc)
def backtrack3(dp,row,col,ans1,ans2):
    #print(str(row)+" "+str(col)+" "+str(dp[row][col]))
    if dp[row][col]==0:
        #print(str(row)+" "+str(col))
        print("Score: " +str(dp[maxrow][maxcol]))
        print(ans1)
        for i in range(len(ans1)):
            if ans1[i]==ans2[i]:
                print("|",end="")
```

```
            else:
                print(" ",end="")
        print()
        print(ans2)
        print()
        return
    if(s1[row-1]!=s2[col-1]):
        if dp[row][col] == dp[row-1][col-1]+mismatch:
            backtrack3(dp,row-1,col-1,s1[row-1]+ans1,s2[col-1]+ans2)
        if dp[row][col] == dp[row-1][col]+gap:
            backtrack3(dp,row-1,col,s1[row-1]+ans1,"-"+ans2)
        if dp[row][col] == dp[row][col-1]+gap:
            backtrack3(dp,row,col-1,"-"+ans1,s2[col-1]+ans2)

    else:
        backtrack3(dp,row-1,col-1,s1[row-1]+ans1,s2[col-1]+ans2)
        if dp[row][col] == dp[row-1][col]+gap:
            backtrack3(dp,row-1,col,s1[row-1]+ans1,"-"+ans2)
        if dp[row][col] == dp[row][col-1]+gap:
            backtrack3(dp,row,col-1,"-"+ans1,s2[col-1]+ans2)

print("Optimal alingments - ")
print()
for i in maxloc:
    backtrack3(dp2,i[0],i[1],"","")
```

```
The DP Matrix will be-
 0  0  0  0  0  0  0  0
 0  0  0  0  2  0  0  2
 0  2  2  0  0  1  2  0
 0  0  1  4  2  0  0  1
 0  0  0  2  6  4  2  2
 0  0  0  0  4  8  6  4
 0  0  0  0  2  6  7  8
 0  0  0  0  0  4  5  6
 0  2  2  0  0  2  6  4
 0  0  1  1  2  0  4  8

Optimal alingments -

Score: 8
TCAG
||||
TCAG

Score: 8
TCAG-A
```

```
|||| |
TCAGTA

Score: 8
AGTA
||||
AGTA

Score: 8
TCAGAGTA
|||  |||
TCA--GTA

Score: 8
TCAGAGTA
||||  ||
TCAG--TA
```