CMSC 423: Solutions to Homework 1

Solution 1:

(a) The dynamic programming table for the problem is shown below. Optimal alignment is

TACGGGTAT
GGAC_G_TAGG

and its cost is 1.

	_	T	A	C	G	G	G	T	A	T
_	0	1	2	3	4	5	6	7	8	9
G	1	1	2	3	2	3	4	5	6	7
G	2	2	2	3	2	1	2	3	4	5
A	3	3	1	2	3	2	2	3	2	3
C	4	4	2	0	1	2	3	3	3	3
G	5	5	3	1	-1	0	1	2	3	4
T	6	4	4	2	0	0	1	0	1	2
A	7	5	3	3	1	1	1	1	-1	0
C	8	6	4	2	2	2	2	2	0	0
G	9	9	7	5	3	1	1	2	1	1

Arrows are:

	_	T	A	C	G	G	G	T	A	T
_										
G		↖								
G		↑								
A			↖							
C				↖	→					
G						↖	→			
T								↖		
A									↖	
C										↖
G										↑

(b) The dynamic programming table for the problem is shown below. Optimal local alignment is

TACG
TAGG

and its cost is -4.

	_	T	A	C	G	G	G	T	A	T
_	0	0	0	0	0	0	0	0	0	0
G	0	0	0	-1	-1	-1	-1	0	0	0
G	0	0	0	0	-1	-2	-2	-1	0	0
A	0	0	-1	0	0	-1	-1	-1	-2	-1
C	0	0	0	-2	-1	0	0	0	-1	-1
G	0	0	0	0	-3	-2	-1	0	0	0
T	0	-1	0	0	-2	-2	-1	-2	-1	-1
A	0	0	-2	-1	-1	-1	-1	-1	-3	-2
C	0	0	0	-3	-2	-1	0	0	-2	-2
G	0	0	0	-2	-4	-3	-2	-1	-1	-1

Some of the arrows are:

	_	T	A	C	G	G	G	T	A	T
_										
G										
G										
A										
C										
G										
T		↖								
A			↖							
C				↖						
G					↖					

(c) There will be 3 dynamic programming tables such as M, X, Y. The optimal alignment will be

```
TACGGGTAT
GGACGTAGG
```

and its cost is 7.

Since gap_{start} is 20 and it is a lot larger than gap_{extend} and other costs, putting even 1 gap will be costlier than aligning both strings directly. Therefore, optimal alignment is as shown above can be found without even filling the tables in this question. However, the question asks for the tables to be filled in:

```
M matrix =
          *     T     A     C     G     G     G     T     A     T
    *     0  -inf  -inf  -inf  -inf  -inf  -inf  -inf  -inf  -inf
    G  -inf    -1   -22   -23   -22   -23   -24   -27   -28   -29
    G  -inf   -22    -2   -23   -22   -21   -22   -25   -28   -29
    A  -inf   -23   -21    -3   -24   -23   -22   -23   -24   -29
    C  -inf   -24   -24   -20    -4   -25   -24   -23   -24   -25
    G  -inf   -25   -25   -25   -19    -3   -24   -25   -24   -25
```

```
    T  -inf   -24   -26   -26   -26   -20    -4   -23   -26   -23
    A  -inf   -27   -23   -27   -27   -27   -21    -5   -22   -27
    C  -inf   -28   -28   -22   -28   -28   -26   -22    -6   -23
    G  -inf   -29   -29   -29   -21   -27   -25   -27   -23    -7
X matrix =
          *     T     A     C     G     G     G     T     A     T
    *     0  -inf  -inf  -inf  -inf  -inf  -inf  -inf  -inf  -inf
    G   -21   -42   -43   -44   -45   -46   -47   -48   -49   -50
    G   -22   -22   -43   -44   -43   -44   -45   -48   -49   -50
    A   -23   -23   -23   -44   -43   -42   -43   -46   -49   -50
    C   -24   -24   -24   -24   -44   -43   -43   -44   -45   -50
    G   -25   -25   -25   -25   -25   -44   -44   -44   -45   -46
    T   -26   -26   -26   -26   -26   -24   -45   -45   -45   -46
    A   -27   -27   -27   -27   -27   -25   -25   -44   -46   -44
    C   -28   -28   -28   -28   -28   -26   -26   -26   -43   -45
    G   -29   -29   -29   -29   -29   -27   -27   -27   -27   -44
Y matrix =
          *     T     A     C     G     G     G     T     A     T
    *     0   -21   -22   -23   -24   -25   -26   -27   -28   -29
    G  -inf   -42   -22   -23   -24   -25   -26   -27   -28   -29
    G  -inf   -43   -43   -23   -24   -25   -26   -27   -28   -29
    A  -inf   -44   -44   -42   -24   -25   -26   -27   -28   -29
    C  -inf   -45   -45   -45   -41   -25   -26   -27   -28   -29
    G  -inf   -46   -46   -46   -46   -40   -24   -25   -26   -27
    T  -inf   -47   -45   -46   -47   -47   -41   -25   -26   -27
    A  -inf   -48   -48   -44   -45   -46   -46   -42   -26   -27
    C  -inf   -49   -49   -49   -43   -44   -45   -46   -43   -27
    G  -inf   -50   -50   -50   -50   -42   -43   -44   -45   -44
```

Solution 2:

The most direct route to a solution is to write a dynamic programming recurrence for the problem. Let $\mathrm{SCS}(i, j)$ be the length of a shortest common supersequence of $X[1..i]$ and $Y[1..j]$. It can be computed as below:

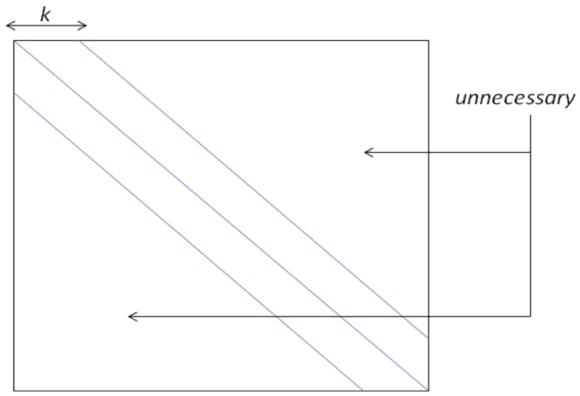
$$\mathrm{SCS}(i,j) = \min \begin{cases} j & \text{if } i = 0 \\ i & \text{if } j = 0 \\ \mathrm{SCS}(i-1, j-1) + 1 & i, j > 0 \text{ and } x_i = y_j \\ \mathrm{SCS}(i, j-1) + 1 & i, j > 0 \text{ and } x_i \neq y_j \\ \mathrm{SCS}(i-1, j) + 1 & i, j > 0 \text{ and } x_i \neq y_j \end{cases} \tag{1}$$

We fill in a table where entry (i, j) has the value of $\mathrm{SCS}(i, j)$ by starting from small values of i and j and working towards larger values. If we trace backwards vertically or horizontally, we add the character in the way we moved to our shortest common supersequence (scs). If we move diagonally, the characters must be the same and we add one of them to scs.

Alternative solution: You can notice that the shortest common supersequence between x and y can be found from an alignment between x and y with a scoring function that rewards matches and does not allow mismatches. E.g.: match $= -1$, mismatch $= \infty$, gap $= 0$. This will find the alignment that has as many matches as possible and no mismatches. The characters in the columns of the alignment will be the shortest common supersequence.

Solution 3:

The main idea is that whole matrix won't be necessary in this case. Only the squares with distance less than or equal to k from the diagonal will be relevant (either above or below the diagonal) in addition to diagonal itself. Thus we're only filling the $(2k+1)n$ squares. The figure above shows this .



We then run the dynamic programming algorithm by considering only these squares and not filling in the others.

Note that the problem was phrased as the user *promising* that the optimal had fewer than k gap pairs. A solution that stays entirely within the band of width k around the diagonal can still have more than k gaps. Since we were promised this won't happen in our case, we are sure to find an optimal solution with $< k$ gap pairs.

Solution 4:

Here is implementation of the program in Python:

```
PROFIT = {('C','c'): 75, ('C','w'): 50, ('W', 'c'): 50, ('W', 'w'): 100}
ADCOST = 10
SWITCHCOST = 25

def bestschedule(weather):
    print 'ADCOST=%d SWITCHCOST=%d' % (ADCOST, SWITCHCOST)
    # D[(A, i)] = the best profit for days 0...i assuming the
    # ad type A was run on day i.
    D = {}

    # A[(a, i)] = the arrow to the i-1 day
    A = {}
```

```
# for day 0, this is the profit:
D[('C', 0)] = PROFIT[('C', weather[0])] - ADCOST
D[('W', 0)] = PROFIT[('W', weather[0])] - ADCOST

for i in xrange(1,len(weather)):
    D[('C', i)] = max(
        D[('C', i-1)],
        D[('W', i-1)] - SWITCHCOST
        ) + PROFIT[('C', weather[i])] - ADCOST

    A[('C', i)] = 'C' if D[('C', i-1)] >= D[('W', i-1)]-SWITCHCOST else 'W'

    D[('W', i)] = max(
        D[('C', i-1)] - SWITCHCOST,
        D[('W', i-1)]
        ) + PROFIT[('W', weather[i])] - ADCOST

    A[('W', i)] = 'W' if D[('W', i-1)] >= D[('C', i-1)]-SWITCHCOST else 'C'

cur = 'C' if D[('C', len(weather)-1)] > D[('W', len(weather)-1)] else 'W'
s = ""
for i in xrange(len(weather)-1, 0, -1):
    s = cur + s
    cur = A[cur, i]
s = cur + s
print weather
print s

print 'W:',
for i in xrange(len(weather)):
    print D[('W', i)],
print '\nC:',
for i in xrange(len(weather)):
    print D[('C', i)],

return max(D[('C', len(weather)-1)], D[('W', len(weather)-1)])
```