Devin Parmet, Igli Duro, David Morris
Dr. William Hendrix
COT 4400

Project 2: Report

1. How would you break down the problem of calculating the maximum alignment score for a combination of two sequences seq1 and seq2 (length n and m, respectively) with the sequence target (length n + m) into one or smaller instances of this problem? Your answer should include what subproblems you are evaluating, as well as how you are using the answers to these subproblems to solve the original problem.

In order to break down the problem described in the PDF for Project 2, we had to Model the problem using a recurrence. The recurrence we came up with involved choosing one element at a time from one of the 2 initial sequences (sequence 1 or sequence 2), multiplying the chosen value and the current value in target, then recursively adding the product of the remaining values of sequence 1 or sequence 2 with the remaining values in target. The recurrence is shown below, note that n is the size of seq1 (where i is the index of seq1), m is the size of seq2 (where j is the index of seq2), and n + m is the size of target (where k is the index of target).

$$cos\_sim(seq1, i, seq2, j, target, k) = max \begin{cases} seq1[i] * target[k] + cos\_sim(seq1, i+1, seq2, j, target, k+1), & \text{if } i != n \text{ AND } j != m \\ seq2[j] * target[k] + cos\_sim(seq1, i, seq2, j+1, target, k+1), & \text{if } i != n \text{ AND } j != m \\ seq1[i] * target[k] + cos\_sim(seq1, i+1, seq2, j, target, k+1), & \text{if } i != n \text{ AND } j == m \\ seq2[j] * target[k] + cos\_sim(seq1, i, seq2, j+1, target, k+1), & \text{if } i == n \text{ AND } j != m \end{cases}$$

2. What are the base cases of this recurrence?

Base Case:

cos_sim(seq1, i, seq2, j, target, k) = {0, if k = n+m

The target sequence has size n + m as noted above in question 1. You can clearly see that the index k of the target is incremented by 1 (k+1 is passed as a parameter) in each recursive call. When k reaches the end of target, or is equal to target's size of n+m, recursion should halt and there are no more products to add to the dot product so we should return 0 which is our base case as shown.

3. What data structure would you use to recognize repeated problems? You should describe both the abstract data structure as well as its implementation.

We would use a 3d array and store the return values from our cos_sim function so that if a value in position i,j,k exists that isn't a sentinel value then that gets returned.

4. Give the pseudocode for a memoized dynamic programming algorithm to find the maximum alignment score when combining seq1 and seq2 to align with target.

```
Input: seq1,seq2, and target : three input arrays
Input: n,m : size of seq1 and seq2 respectively (target size is n+m)
Algorithm: cos_sim_Wrapper
cs = Array(n+1, m+1, n+m+1);
Initialize all cells of cs to -2; //this is our sentinel value
return cos_sim(seq1, 0, seq2, 0, target, 0);

Input: seq1, seq2, and target : the three input arrays
Input: i, j, and k: the indexes of seq1, seq2,and target respectively
Algorithm: cos_sim
if(cs[i][j][k] != -2)
        return cs[i][j][k];
if(k == n+m)
        Return 0;
else if(i != n && j != m)
        op1 = seq1[i]*target[k] + cos_sim(seq1, i+1, seq2, j, target, k+1);
        op2 = seq2[j]*target[k] + cos_sim(seq1, i, seq2, j+1, target, k+1);
else if(i == n && j != m)
        op1 = seq2[j]*target[k] + cos_sim(seq1, i , seq2, j+1, target, k+1);
else if(i != n && j == m)
        op2 = seq1[i]*target[k] + cos_sim(seq1, i+1 , seq2, j, target, k+1);
cs[i][j][k] = max(op1, op2);
return cs[i][j][k];
```

5. Give pseudocode for an iterative algorithm to find the maximum alignment score when combining seq1 and seq2 to align with the target. This algorithm does not need to have a reduced space complexity relative to the memoized solution.

```
Input: seq1,seq2, and target : three input arrays
Input: n,m : size of seq1 and seq2 respectively (target size is n+m)
Algorithm: cos_sim_Iter
cs = Array(n+1, m+1, n+m+1);

op1, op2 = 0;


for (i = n; i > 0; i--)
        k = i + j
        for(j = m; j > 0; j--)
                if(i != n && j != m)
                        temp1 = op1;
                        op1 = seq1[i]*target[k];
                        temp1 = temp1 + op1;

                        temp2 = op2;
                        op2 = seq2[j]*target[k];
                        temp2 = temp2 + op2;
                else if(i == n && j != m)
                        temp1 = op1;
                        op1 = seq2[j]*target[k]
                        temp1 = temp1 + op1;
                else if(i != n && j == m)
                        temp2 = op2;
                        op2 = seq1[i]*target[k]
                        temp2  = temp2 + op1;

                cs[i][j][k] = max(op1, op2);
                k = k - 1;
return cs;
```

6. Can the space complexity of the iterative algorithm be improved relative to the memoized algorithm? Justify your answer.
   Yes, the space complexity could be improved. In our memoized implementation we utilized a 3D array based map, when printing out the contents of the 3D structure, there were a lot of untouched sentinel values that occupied a substantial amount of unused space. We chose the

3D structure because it offered the benefit of being simpler to code as we did not have to account for values that would overwrite each other, at the cost of space. A potential solution to this is to instead use a 2D array based map that focuses on storing s-values using only the indexes i and j that correspond to seq1 and seq2 respectively. In this potential solution, we would have to find some other way to keep track of the index k for target when storing values because in each recursive call, the target sequence is incremented so k is always incremented by 1 but this is not the case for i and j because only one element from either seq1 or seq2 is chosen in each recursive call. The main idea in this potential solution is that k is always incremented until it reaches the size of target which is the size of the solution sequence. Since k is always incremented it has a different value in each call so by keeping track of this using a different method than storing it as a dimension would increase space complexity significantly by eliminating the extra dimension to utilize a 2D array based map and keeping track of k with a different method.

7. Give pseudocode for an algorithm that identifies the sequence which will achieve the maximum alignment score. If there is more than one sequence that yields the maximum score, you may return any such sequence. Your algorithm may be iterative or recursive.

```
Input: seq1,seq2, and target : three input arrays
Input: n,m : size of seq1 and seq2 respectively (target size is n+m)

retSequence = Array(n+1, m+1, n+m+1);
path = Array(n+m);

Algorithm: cos_sim_Wrapper
Initialize all cells of cs to -2; //this is our sentinel value
cos_sim(seq1, 0, seq2, 0, target, 0);
return(diagonal(retSequence, path));

Input: seq1, seq2, and target : the three input arrays
Input: i, j, and k: the indexes of seq1, seq2,and target respectively
Algorithm: cos_sim
temp1;
temp2;
```

```
if(k == n+m)
        Return 0;
else if(i != n && j != m)
        op1 = seq1[i]*target[k] + cos_sim(seq1, i+1, seq2, j, target, k+1);
        temp1=seq1[i];
        op2 = seq2[j]*target[k] + cos_sim(seq1, i, seq2, j+1, target, k+1);
        temp2=seq2[j];
else if(i == n && j != m)
        op1 = seq2[j]*target[k] + cos_sim(seq1, i , seq2, j+1, target, k+1);
        temp1=seq2[j];
else if(i != n && j == m)
        op2 = seq1[i]*target[k] + cos_sim(seq1, i+1 , seq2, j, target, k+1);
        temp2=seq1[i];

if(op1 > op2)
        retSequence[i][j][k+1] = temp1;
        path[k]="op1";
else
        retSequence[i][j][k+1] = temp2;
        path[k]="op2";
return max(op1, op2);
```