### Sequence Alignment with Profile HMM Problem

*Align a sequence to a family of sequences using a profile HMM.*

**Input:** A string *x*, a threshold *θ*, a pseudocount *σ*, an alphabet *Σ*, and a multiple alignment *Alignment*.

**Output:** An optimal hidden path emitting *x* in HMM(*Alignment*, *θ*, *σ*).

**Input Format.** The first line of the input contains a string *x*. The second line of the input is “--------” (a delimiter). The third line of the input contains the threshold *θ* and the pseudocount *σ* (space-delimited). The fourth line of the input is “--------” (a delimiter). The fifth line of the input is the list of symbols in the alphabet *Σ* (space-separated). The sixth line of the input is “--------” (a delimiter). The remaining lines are the multiple alignment *Alignment*, with one sequence (containing gaps) on each line.

**Output Format.** An optimal hidden path emitting *x* in HMM(*Alignment*, *θ*, *σ*), where the states of the hidden path are space-delimited.

**Constraints.** |*x*| = 50; 0.2 ≤ *θ* ≤ 0.4; *σ* = 0.01 ; |*Σ*| = 5; 5 ≤ |*Alignment*| ≤ 10; |*Alignment*[*i*]| = 50

**SAMPLE DATASET:**

Input:

AEFDFDC

--------

0.4 0.01

--------

A B C D E F

--------

ACDEFACADF

AFDA---CCF

A--EFD-FDC

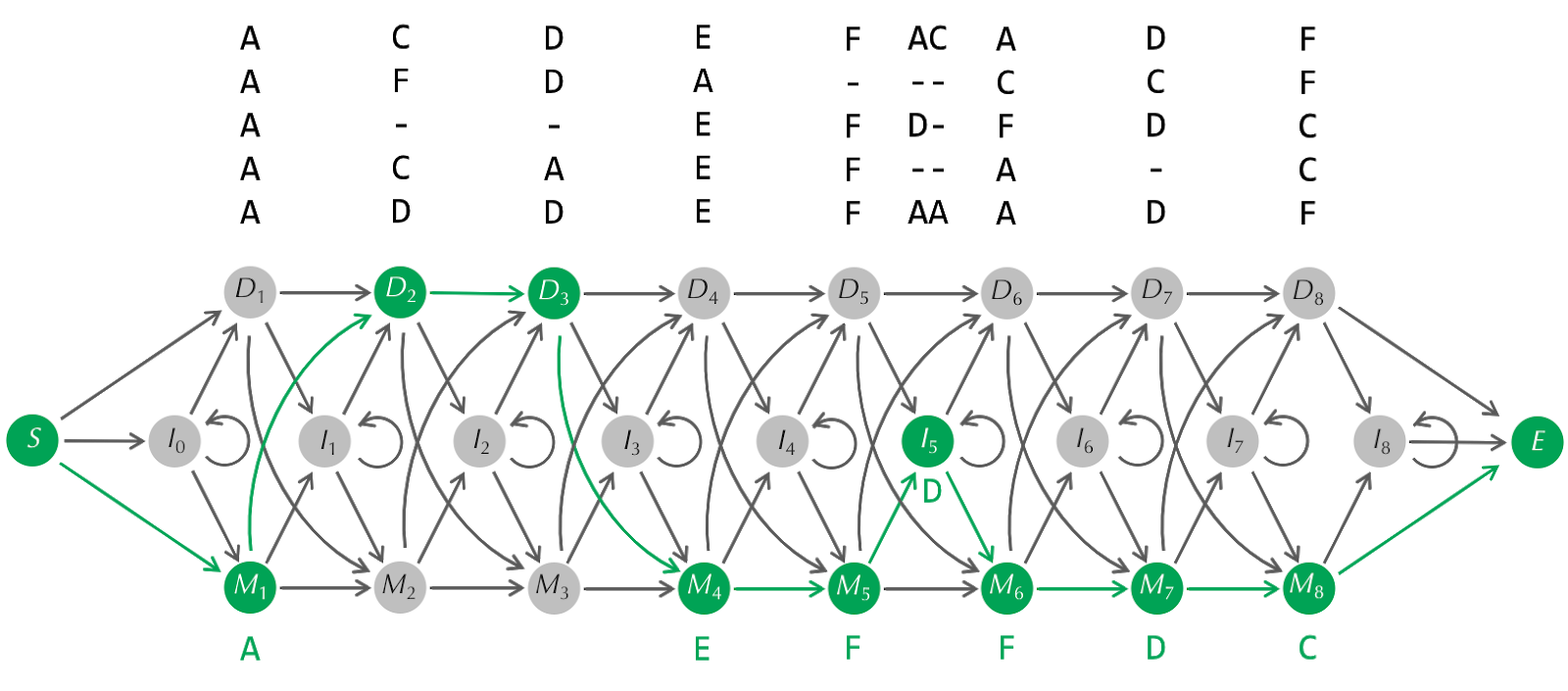
ACAEF--A-C

ADDEFAAADF

Output:

M1 D2 D3 M4 M5 I5 M6 M7 M8

This is an optimal hidden path emitting the given string *x* = AEFDFDC against HMM(*Alignment*, *θ*, *σ*).



**TEST DATASET 1:**

Input:

AB

--------

0.4 0.01

--------

A B

--------

AA

AA

Output:

M1 M2

This dataset makes sure that your code uses match states to represent mismatches as well as matches. The optimal alignment using the profile HMM with pseudocounts consists of a match and a mismatch. If your output does not match the correct output make sure that your code is using a profile HMM with pseudocounts. If your implementation does not use pseudocounts any path through the HMM would have a 0 probability, since the character B is never observed in the original multiple alignment. If your output is incorrect and your code implements a profile HMM with pseudocounts then make sure that your initialization for the Viterbi algorithm is correct. Unlike previous problems in which the Viterbi algorithm was used we do not assume that transitions from the starting state are equally likely. Make sure that your code uses the transition matrix to determine transitions from the starting state.

**TEST DATASET 2:**

Input:

AB

--------

0.4 0.01

--------

A B C D E

--------

AA

AA

Output:

M1 M2

This dataset makes sure that your code is not affected by unused characters in the HMM alphabet. This dataset is the same as Test Dataset 1, except extra characters are added to the alphabet. The unused characters will change the transition and emission matrices for the profile HMM but should not change the final alignment. If your output doesn’t match the correct output make sure that your underlying profile HMM with pseudocounts implementation is correct and unaffected by the additional alignment code added.

**TEST DATASET 3:**

Input:

AB

--------

0.4 0.01

--------

A B

--------

A-

A-

Output:

M1 I1

This dataset makes sure that your code can correctly transition to an insertion state. The second column of the multiple alignment should be excluded from the seed alignment because it contains a higher fraction of gaps that allowed by the threshold. This makes the B character in string *x* an insertion. If your output does not match the correct output make sure that your code does not confuse insertions with deletions. If your code passed Test Dataset 1 then errors in this dataset are likely due to an error in your implementation of a profile HMM with pseudocounts. If your code for the Profile HMM with Pseudocounts problem passed all tests cases make sure that your code for this problem doesn’t invalidate your implementation of the profile HMM with pseudocounts.

**TEST DATASET 4:**

Input:

AAAAAB

--------

0.4 0.01

--------

A B

--------

-B

-B

Output:

I0 I0 I0 I0 I0 M1

This dataset makes sure that your code can handle contiguous runs of insertions. The first column of the multiple alignment is ignored in the seed alignment because it has a higher gap character fraction than the specified threshold. Since the last character of string *x* matches with the B characters in the last column of the multiple alignment the optimal alignment inserts the A characters from string *x* until it can match the B character. If your output doesn’t match the correct output make sure that your code is correctly setting initial probabilities for the first insertion state (I0). Additionally make sure that insertion states can transition to themselves. Make sure that your code passes Test Dataset 2 as well to avoid any initial mistakes with insertion states in the profile HMM alignment.

**TEST DATASET 5:**

Input:

AB

--------

0.4 0.01

--------

A B

--------

AAAAB

AAAAB

Output:

M1 D2 D3 D4 M5

This dataset makes sure that your code is correctly handling deletion states. The regular Viterbi algorithm struggles with silent states, so for profile HMM alignment the silent deletion states need to be handled differently than the non-silent states. Since deletion states should not “consume” any of the characters from string *x* we do not transition into the next column of the dynamic programming matrix at deletion states. Deletion states should only consider states in the same column as candidate previous states. If your code does not take this into account it’s likely that it will crash on this input. If your output is incorrect for this dataset double check your dynamic programming matrix. The matrix should have 2 columns for the 2 characters in string *x* and 16 rows for the 16 possible states (not including start and end states). When filling out the dynamic programming matrix the deletion states should look in the same column when determining the previous state. When backtracking to reconstruct the states of the alignment be sure that you do not switch columns at deletion states. **TEST DATASET 6:**

Input:

B

--------

0.4 0.01

--------

A B

--------

AAAB

AAAB

---B

Output:

D1 D2 D3 M4

This dataset makes sure that your code correctly handles the initial transition. In this dataset the gaps in the bottom string of the multiple alignment increase the probability of transition to a deletion state. Combined with the consensus on a B character in the final column this makes the optimal alignment a run of deletion states ending with a match state. The dynamic programming matrix in this case should only have one column, since string *x* is only one character long. In order to handle deletions at the beginning of the alignment we use set the initial state to a column containing the start state S and all deletion states. If your output doesn’t match the correct output it’s likely that your code does not use this collection of initial states. It is also possible that your code fails to consider these initial states when reconstructing the states. Check that your code correctly handles the initial state.