# Homework 4

Elliott Pryor

14 November 2021

**Problem 1** Consider the follwing set of sequences:

$$S_1 = ACTCTCGATC$$
$$S_2 = ACTTCGATC$$
$$S_3 = ACTCTCTATC$$
$$S_4 = ACTCTCTAATC$$

Compute the MSA using the center star method.

We give the pairwise distances

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $S_1$ | 0 | 1 | 1 | 2 |
| $S_2$ | 1 | 0 | 2 | 3 |
| $S_3$ | 1 | 2 | 0 | 1 |
| $S_4$ | 2 | 3 | 1 | 0 |

We get that $S_1$ or $S_3$ would be equally good center stars. We pick $S_1$.

We compute alignments with $S_1$:

$$\begin{aligned} S_1 &= ACTCTCGATC \\ &= ACTCTCGATC \\ S_2 &= ACTCTCGATC \\ &= ACT\_TCGATC \\ S_3 &= ACTCTCGATC \\ &= ACTCTCTATC \\ S_4 &= ACTCTCG\_ATC \\ &= ACTCTCTAATC \end{aligned}$$

From this we get the MSA:

$$\begin{aligned} &ACTCTCG\_ATC \\ &ACT\_TCG\_ATC \\ &ACTCTCT\_ATC \\ &ACTCTCTAATC \end{aligned}$$

**Problem 2** Compute the ClustalW alignment of the previous problem:

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $S_1$ | 0 | 0 | 0.1 | 0.1 |
| $S_2$ | 0 | 0 | $\bar{0.1}$ | $\bar{0.1}$ |
| $S_3$ | 0 | 0 | 0 | 0 |
| $S_4$ | 0 | 0 | 0 | 0 |

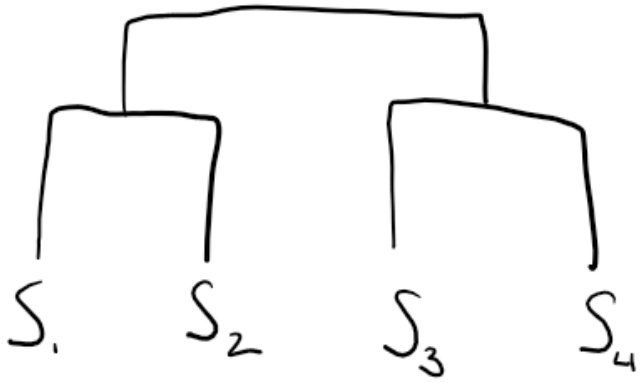


Figure 1: Guide tree

Aligning $S_1, S_2$ produces $(ACTCTCGATC, ACT\_TCGATC)$,

aligning $S_3, S_4$ produces $(ACTCTCT\_ATC, ACTCTCTAATC)$.

Then using the profile-profile alignment we get:

$$
\begin{array}{l}
ACTCTCGATC\_ \\
ACT\_TCGATC\_ \\
ACTCTCTATC\_ \\
ACTCTCTAATC
\end{array}
$$

**Problem 3** Given a set of $k$ sequences: $S_1, \ldots, S_k$ find $k$ substrings such that the optimal SP score of multiple sequence alignment is maximized. Give a Dynamic Programming algorithm to solve this. (Note, when $k = 2$ this is same as solving local alignment problem)

We build a generalized version of Smith-Waterman algorithm. We base this on the generalized verson of Needleman-Wunch algorithm from class:

$$V(i_1, \ldots, i_k) = \max_{b_1,\ldots,b_k \in \{(0,1)\}^k \setminus (0,0)} V(i_1 - b_1, \ldots, i_k - b_k) + SP(S_1[i_1 b_1], \ldots S_k[i_k b_k])$$

We note that the only difference in the Smith-Waterman algorith from the Needleman-Wunch algorithm is the option to take a score of 0 always. So we have a generalized Smith-Waterman:

$$V(i_1, \ldots, i_k) = \max_{b_1,\ldots,b_k \in \{(0,1)\}^k \setminus (0,0)} (V(i_1 - b_1, \ldots, i_k - b_k) + SP(S_1[i_1 b_1], \ldots S_k[i_k b_k]), 0)$$

With a base case of $V(0, \ldots, 0) = 0$ and $i_j = 0 \implies b_j = 0$.

Then we have to find the local alignment with the best score. This is the entry with the greatest value in the matrix. So the optimal score is $max_{i_1 \ldots i_k} V(i_1 \ldots i_k)$.

Building the matrix $V$ takes $O(n^k)$ space. For running time we have $O(n^k)$ entries to fill out, $2^k - 1$ options for $b$, and $\binom{k}{2}$ evaluations of $SP$. Thus we have $O(n^k 2^k k^2)$ time (same as generalized Needleman-Wunch) to build $V$. Then we need to find the max in $V$: which is just searching through all $n^k$ entries. Thus our running time is: $O(n^k 2^k k^2)$

**Problem 4** Code Center Star Algorithm.

```
PS D:\pryor\Documents\GitHubProjects\CompBio\HW4> python .\Center_Star.py 1 1 example1.fasta
Opening: example1.fasta
Center (string, index): ('bananan', 0)
MSA:
banana_n_
ba__n__n_
aannna_na
_ananan_a
banana_n_
```

Figure 2: Example showing execution on example1

```
PS D:\pryor\Documents\GitHubProjects\CompBio\HW4> python .\Center_Star.py 1 1 example2.fasta
Opening: example2.fasta
Center (string, index): ('ACTCTCGATC', 0)
MSA:
ACTCTCG_ATC
ACT_TCG_ATC
ACTCTCT_ATC
ACTCTCTAATC
```

Figure 3: Example showing execution on strings from Problem 1

> EXAMPLE 1 seq 1
bananan
> seq 2
bann
> seq 3
aannnana
> seq 4
ananana
> seq 5
bananan

> EXAMPLE 2 seq1
ACTCTCGATC
>seq2
ACTTCGATC
>seq3
ACTCTCTATC
>seq4
ACTCTCTAATC

```python
from EditDistance import *
import numpy as np


def compute_indx(Sc,s2):
    indx = []
    for i in range(len(s2)):
        if i >= len(Sc) or (s2[i] == '_' and Sc[i] != '_'):
            indx.append(i)
            Sc = Sc[:i] + '_' + Sc[i:]
```

```
    indx += [len(Sc) + i for i in range(len(s2) - len(Sc))]
    return indx


def center_star(strings, f=delta):
    """
    :param strings: List of strings to compute
    :return: center string, index of center string in strings array (param)
    """

    mat = np.zeros((len(strings), len(strings)))  # make k x k matrix
    for i, S in enumerate(strings):
        for j, T in enumerate(strings):
            if j <= i:  # skip the lower diagonal elements
                continue
            D = needleman(S, T)[0]
            mat[i][j] = D[len(S)][len(T)]  # get the edit distance
            mat[j][i] = mat[i][j]

    # the string that minimizes sum of distances to all other strings
    center = np.argmin(mat.sum(axis=1))
    return strings[center], center


def MSA(strings, f=delta):
    Sc, center = center_star(strings)
    alignment = [s for s in strings]  # copy the input strings
    for i in range(len(strings)):
        if i == center:  # skip center string
            continue

        V, P = needleman(Sc, strings[i], f)
        s2, t2 = construct_alignment(P, Sc, strings[i])
        alignment[i] = t2

        add_spaces = compute_indx(Sc, s2)
        for indx in add_spaces:
            for j in range(i):
                alignment[j] = alignment[j][:indx] + '_' + alignment[j][indx:]
        Sc = s2
    alignment[center] = Sc

    return alignment


def read_fasta(filepath):
    """
    Returns a dict key=sequence_name, value=sequence from the fasta file
    :param filepath:
    :return:
    """
    file = open(filepath)
    print(f"Opening: {filepath}")
    started = False
    header = ''
```

```
    text = ''
    sequences = {}
    for line in file:  # reads in fasta file. Allows for sequence to continue on multiple lines
        if line.startswith('>'):
            if started:
                sequences[header] = text.replace('\n', '')
                text = ''
            started = True
            header = line.replace('>', '').strip()
        else:
            text += line.strip()
    sequences[header] = text.replace('\n', '')
    return sequences


def diff_generator(a, b):
    def f(c1, c2):
        # Helper function used for scoring
        if c1 == c2:
            return 0
        elif c1 == '_' or c2 == '_':
            return b
        else:
            return a
    return f


if __name__ == "__main__":
    if len(sys.argv) == 4:
        alpha = float(sys.argv[1])
        beta = float(sys.argv[2])
        file = sys.argv[3]
        seq = read_fasta(file)
        f = diff_generator(alpha, beta)
        ins = list(seq.values())

        print(f'Center (string, index): {center_star(ins, f)}')
        print("MSA:")
        msa = MSA(ins, f)
        for s in msa:
            print(s)
    else:
        print(center_star(['ACTCTCGATC', 'ACTTCGATC', 'ACTCTCTATC', 'ACTCTCTAATC']))
        print(center_star(['bananan', 'bann', 'aannnana', 'ananana', 'bananan']))
        msa = MSA(['bananan', 'bann', 'aannnana', 'ananana', 'bananan'])
        for s in msa:
            print(s)
```