Analysis

Time & Space Complexities

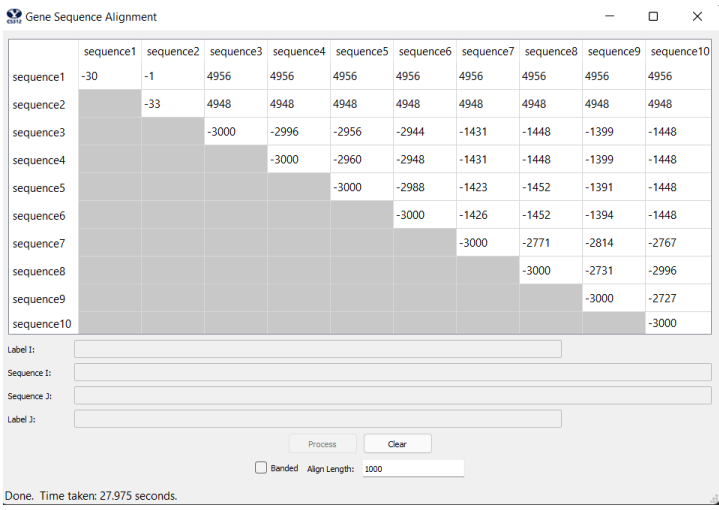
Unrestricted Algorithm. The time complexity for the unrestricted algorithm is O(*nm*) because the outer loop in the double for-loop iterates *n* times, and the inner loop iterates *m* times. There are a few loops before the double loop, but each is only O(*n*) or O(*m*). Besides those, all other operations are O(1). The space complexity for the unrestricted algorithm is also O(*nm*) because it creates two matrices of size *n* x *m*.

Unrestricted Algorithm. The time complexity for the banded algorithm is O(*kn*) because the outer loop in the double for-loop iterates *n* times, and the inner loop iterates 2*d*+1 = *k* times. There is a loop after the double loop, but it is only O(*k*). Besides that, all other operations are O(1). The space complexity for the banded algorithm is also O(*kn*) because it creates two matrices of size *k* x *n*.

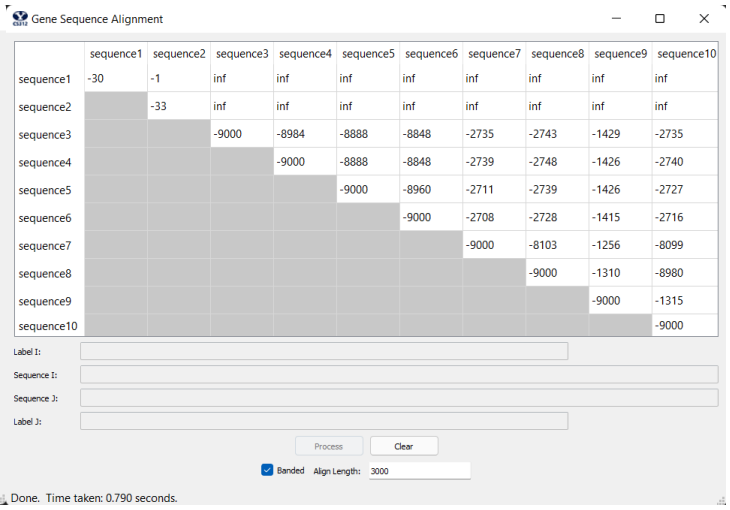
Alignment Extraction Algorithm

My alignment extraction algorithm takes the two sequences being aligned and a matrix of back-pointers (P) that was created in either of the edit distance methods. P has the same dimensions as the distance matrix. Each entry in P represents whether an insertion, deletion or substitution was made to get to that position. A 0 represents a substitution (or match), a 1 represents an insertion, and a 2 represents a deletion. The algorithm starts by setting indices *i* and *j* to the final position in the matrix (bottom right for unrestricted, just before the first infinity of the last row for banded). It then moves opposite the direction of the respective alignment (i.e., if the alignment is a substitution, then *i* and *j* increment by 1 in the unrestricted algorithm, and *i* increments by 1 in the banded algorithm, so for the alignment extraction algorithm *i* and *j* will decrement by 1 if unrestricted, and *i* will decrement in the banded algorithm). Two new empty strings are created for the seq1_aligned and seq2_aligned. Then we will work backwards through seq1 and seq2. If the entry at index (*i*,*j*) is an insertion, "-" is prepended to seq1_aligned and the character at the current position of seq2 is prepended to seq2_aligned. If the entry at index (*i*,*j*) is a deletion, "-" is prepended to seq2_aligned and the character at the current position of seq1 is prepended to seq1_aligned. If the entry at index (*i*,*j*) is an substitution, the character at the current position of seq1 and seq2 are prepended to seq1_aligned and seq2_aligned, respectively. The first 100 character of seq1_aligned and seq2_aligned are returned.

Results



	sequence1	sequence2	sequence3	sequence4	sequence5	sequence6	sequence7	sequence8	sequence9	sequence10
sequence1	-30	-1	4956	4956	4956	4956	4956	4956	4956	4956
sequence2		-33	4948	4948	4948	4948	4948	4948	4948	4948
sequence3			-3000	-2996	-2956	-2944	-1431	-1448	-1399	-1448
sequence4				-3000	-2960	-2948	-1431	-1448	-1399	-1448
sequence5					-3000	-2988	-1423	-1452	-1391	-1448
sequence6						-3000	-1426	-1452	-1394	-1448
sequence7							-3000	-2771	-2814	-2767
sequence8								-3000	-2731	-2996
sequence9									-3000	-2727
sequence10										-3000



	sequence1	sequence2	sequence3	sequence4	sequence5	sequence6	sequence7	sequence8	sequence9	sequence10
sequence1	-30	-1	inf	inf	inf	inf	inf	inf	inf	inf
sequence2		-33	inf	inf	inf	inf	inf	inf	inf	inf
sequence3			-9000	-8984	-8888	-8848	-2735	-2743	-1429	-2735
sequence4				-9000	-8888	-8848	-2739	-2748	-1426	-2740
sequence5					-9000	-8960	-2711	-2739	-1426	-2727
sequence6						-9000	-2708	-2728	-1415	-2716
sequence7							-9000	-8103	-1256	-8099
sequence8								-9000	-1310	-8980
sequence9									-9000	-1315
sequence10										-9000

Unrestricted

seq3

```
gattgcgagcgatttgcgtgcgtgcatcccgcttc-actg--at-ctctt
```

```
gttagatcttttcataatctaaactttataaaacatccactccctgta-
```

seq10

```
aataa-gagtgattggcgtccgtacgtacccttttctactctaaactcttg
```

```
ttagtttaaatc-taatctaaactttataaa--cggc-acttcctgtgt
```

Banded

seq3

```
gattgcgagcgatttgcgtgcgtgcatcccgcttc-actg--at-ctctt
```

```
gttagatcttttcataatctaaactttataaaacatccactccctgta-
```

seq10

```
aataa-gagtgattggcgtccgtacgtacccttttctactctaaactcttg
```

```
ttagtttaaatc-taatctaaactttataaa--cggc-acttcctgtgt
```

Source Code

```
class GeneSequencing:

    def __init__(self):
        pass


    def align(self, seq1, seq2, banded, align_length):
        self.banded = banded
        self.MaxCharactersToAlign = align_length

        score, P = self.getEditDistance(seq1, seq2)
        alignment1, alignment2 = self.aligned(seq1, seq2, P)

        return {'align_cost': score, 'seqi_first100': alignment1, 'seqj_first100':
alignment2}

    # takes two sequences as arguments
    # returns banded edit distance if self.banded is true; otherwise, returns
unrestricted edit distance
    def getEditDistance(self, seq1, seq2):
        if self.banded:
            return self.getEditDistanceBanded(seq1, seq2)
        else:
            return self.getEditDistanceUnrestricted(seq1, seq2)

    #  Edit Distance Algorithm (Banded Implementation)
    #  takes two sequences as arguments
    #  returns banded edit distance and 2D array of back-pointers
    #  t: O(kn)  outer loop of double for-loop iterates n times. inner for-loop iterates
k times.
    #  s: O(kn)  two 2-dimensional arrays (E & P) of size n x k are created.
```

```
    def getEditDistanceBanded(self, seq1, seq2):
        seq1 = seq1[:self.MaxCharactersToAlign]
        seq2 = seq2[:self.MaxCharactersToAlign]
        len1 = len(seq1) + 1  # n = len1 = len(seq1) (or align length)
        len2 = 2 * MAXINDELS + 1  # k = len2 = 2d + 1
        if abs(len(seq1) - len(seq2)) > MAXINDELS:
            return math.inf, []

        E = []
        P = []
        for i in range(len1):  # t: O(kn)  s: O(kn)
            E.append([])
            P.append([])
            for j in range(len2):
                if len(seq2) >= i + j - MAXINDELS >= 0:
                    if i == 0:
                        E[i].append(5 * (j - MAXINDELS))
                        P[i].append(1)
                    else:
                        if j == len2 - 1:
                            e = min(self.diff(seq1[i - 1], seq2[i + j - MAXINDELS - 1]) +
E[i - 1][j], INDEL + E[i][j - 1])
                            if e == INDEL + E[i][j - 1]:
                                P[i].append(1)
                            else:
                                P[i].append(0)
                        elif j == 0:
                            e = min(self.diff(seq1[i - 1], seq2[i + j - MAXINDELS - 1]) +
E[i - 1][j],
                                    INDEL + E[i - 1][j + 1])
                            if e == INDEL + E[i - 1][j + 1]:
                                P[i].append(2)
                            else:
                                P[i].append(0)
                        else:
                            e = min(self.diff(seq1[i - 1], seq2[i + j - MAXINDELS - 1]) +
E[i - 1][j], INDEL + E[i][j - 1],
                                    INDEL + E[i - 1][j + 1])
                            if e == INDEL + E[i][j - 1]:
                                P[i].append(1)
                            elif e == INDEL + E[i - 1][j + 1]:
                                P[i].append(2)
                            else:
                                P[i].append(0)
                        E[i].append(e)
                else:
                    E[i].append(math.inf)
                    P[i].append(math.inf)
        ret_j = len2 - 1
        ret = E[len1 - 1][ret_j]
        while ret == math.inf:  # t: O(k)  s: O(1)
            ret_j -= 1
            ret = E[len1 - 1][ret_j]
        return ret, P

    #  Edit Distance Algorithm (Unrestricted Implementation)
    #  takes two sequences as arguments
    #  returns unrestricted edit distance and 2D array of back-pointers
    #  t: O(nm)  outer loop of double for-loop iterates n times. inner for-loop iterates
m times.
    #  s: O(nm)  two 2-dimensional arrays (E & P) of size n x m are created.
    def getEditDistanceUnrestricted(self, seq1, seq2):
        len1 = min(len(seq1), self.MaxCharactersToAlign) + 1  # n = len1 = len(seq1) (or
```

```
align length)
        len2 = min(len(seq2), self.MaxCharactersToAlign) + 1  # m = len2 = len(seq2) (or
align length)
        E = []
        P = []
        for i in range(len1):  # t: O(n)  s: O(2n)
            E.append([5 * i])
            P.append([0])
        for j in range(1, len2):  # t: O(m)  s: O(2m)
            E[0].append(5 * j)
            P[0].append(1)
        for i in range(1, len1):  # t: O(nm)  s: O(2nm)
            for j in range(1, len2):
                e = min(self.diff(seq1[i - 1], seq2[j - 1]) + E[i - 1][j - 1], INDEL +
E[i][j - 1], INDEL + E[i - 1][j])
                E[i].append(e)
                if e == INDEL + E[i][j - 1]:
                    P[i].append(1)
                elif e == INDEL + E[i - 1][j]:
                    P[i].append(2)
                else:
                    P[i].append(0)
        return E[len1 - 1][len2 - 1], P

    #  takes two characters as arguments
    #  returns MATCH if characters match; otherwise, returns SUB
    def diff(self, i, j):  # t: O(1)
        if i == j:
            return MATCH
        else:
            return SUB

    #  takes two sequences and a 2D array of back-pointers as arguments
    #  returns first 100 characters of each sequence aligned using the back-pointers
    def aligned(self, seq1, seq2, P):
        if not P:
            return "No Alignment Possible", "No Alignment Possible"
        if self.banded:
            return self.alignedBanded(seq1, seq2, P)
        else:
            return self.alignedUnrestricted(seq1, seq2, P)

    #  Alignment Extraction Algorithm (Banded Implementation)
    #  takes two sequences and a 2D array of back-pointers as arguments
    #  returns first 100 characters of each sequence aligned using the back-pointers
    def alignedBanded(self, seq1, seq2, P):
        i = len(P) - 1
        j = len(P[i]) - 1
        while P[i][j] == math.inf:
            j -= 1

        seq1_aligned = seq2_aligned = ''

        while not (i == 0 and j == MAXINDELS):
            if P[i][j] == 2:
                seq1_aligned = seq1[i - 1] + seq1_aligned
                seq2_aligned = "-" + seq2_aligned
                i -= 1
                j += 1
            elif P[i][j] == 0:
                seq1_aligned = seq1[i - 1] + seq1_aligned
                seq2_aligned = seq2[i + j - MAXINDELS - 1] + seq2_aligned
                i -= 1
```

```
        else:
            seq1_aligned = "-" + seq1_aligned
            seq2_aligned = seq2[i + j - MAXINDELS - 1] + seq2_aligned
            j -= 1

    return seq1_aligned[:100], seq2_aligned[:100]

#  Alignment Extraction Algorithm (Unrestricted Implementation)
#  takes two sequences and a 2D array of back-pointers as arguments
#  returns first 100 characters of each sequence aligned using the back-pointers
def alignedUnrestricted(self, seq1, seq2, P):
    i = len(P) - 1
    j = len(P[i]) - 1

    seq1_aligned = seq2_aligned = ''

    while i > 0 or j > 0:
        if P[i][j] == 2:
            seq1_aligned = seq1[i - 1] + seq1_aligned
            seq2_aligned = "-" + seq2_aligned
            i -= 1
        elif P[i][j] == 0:
            seq1_aligned = seq1[i - 1] + seq1_aligned
            seq2_aligned = seq2[j - 1] + seq2_aligned
            i -= 1
            j -= 1
        else:
            seq1_aligned = "-" + seq1_aligned
            seq2_aligned = seq2[j - 1] + seq2_aligned
            j -= 1

    return seq1_aligned[:100], seq2_aligned[:100]
```