Katherine Filpo Lopez
Professor Kerri-Ann Norton
October 23rd 2020
Lab Report #5: Sequence Variation and Alignment

Introduction

In order to compare many sequences with many possible origins, one has to be able to align them in order to analyze the sequences. In this lab, we are modeling the different ways that sequences can be mutated and then comparing them to each other, using the multiple alignment algorithm. This lab's focus was more on making the mutations happen in many different ways instead of to align them.

We were simulating mistakes that would naturally occur in the cell due to many possible reasons. A real life example of this is cancer cells. When a cell in the human body gets too many mutations because of sun radiation or age, the cell can mutate in such a way that it becomes malicious to the body. One can study the mutated sequences and compare them to the original sequence and see where the genes broke and what caused a problem in the cell to make them malicious to the body. It is very important to be able to track these changes as it can be important when creating methods to treat this ailment.

Methods

The code has two major classes, Mutation and SequenceVariationDetecion, and import functions from the MultipleAlignment file. The seven sequences that we are comparing are called outside of these two classes. They are made by calling functions of the Mutation class and storing the results in the sequence variable that is made. Everytime they are called with the intention to get the score or to align them, it's always different because the mutations that happen to the sequence are always changing every time the program is run.

The classes are separated into two because the Mutation class is making many changes to the sequence, and doing things like mutating transposons and looking for motifs to delete or add another copy, while the SequenceVariationDetecion class just looks for the mutations that were made by the Mutation class. This mutation detection class uses two sequences, the original and the second possible mutated one and compares to see what kind of mutation it is. Here is an example of the Mutation class methods:

```
def crossoverMutation(self, seq2):
        sequenceList1 = list(self.getSeq())
        sequenceList2 = list(seq2)
        placeholderlist = sequenceList1
        mutationSite = random.randint(0,len(sequenceList1)-1)
        sequenceList1 = sequenceList1[:mutationSite] + sequenceList2[mutationSite+1:]
        sequenceList2 = sequenceList2[:mutationSite] + placeholderlist[mutationSite+1:]
        return ''.join(sequenceList1), ''.join(sequenceList2)
```

All the methods are similar in the sense that they use lists made in the beginning of the method to move the sequence around in some way that simulates mutations. This example is for crossovers of sequences.

The MultipleAlignment file that is imported is written by Professor Kerri-Ann and works very well. All I have to do is use the imported methods to get the score of the sequence and to align the multiple sequences for me.

Results

The mutation class works very well. It gives one the mutated sequences, so it does what it's supposed to. The methods for the most part work well and give the expected results.

The method named transposon, might be a little wrong. It inserts the transposon sequence well, the issue is that it cannot find the transposon part of the sequence and needs a motif to guide it. Also, while theSequenceVariationDetecion class seems to work as well as the Mutation class, there might be some issues in the point mutation method, as I call the crossover detection method in there because the crossover of the sequences will mean that they have the same length, and could possibly be another point mutation.

Here are some such results of what one gets when running the mode and aligning the sequence using the MultipleSequence algorithm:

```
[25.0, -27.0, -28.0, -32.0, -19.0, -23.0, -27.0]
0 ------------GCA--CGT--ATT-GATTGGCC-TGTACCTA
1 ---------------T-CTTGAATTACATAGCCCGGGT-C-GA
2 ----------------GC-AGAATTACATAGCCCGGGT-C-GA
3 ---------------T-CTTGAATTACATAGCCCGGGT-T--A
4 ----------GCACGTACGT-A-TT-GATTGGCC-TGTACCTA
```

```
5 --------------------GCATT-GATTGGCC-TGTACCTA
6 GCACGTGGTTGCACGTACGT-A-TT-GATTGGCC-TGTACCTA
None
```

Conclusion

Code works adequately, and I am satisfied with the code.