COMP 555 - Problem Set 1

Joe Puccio

February 25, 2015

1.

We can see that there exist $4^{(6/2)} = 4^3 = 64$ different reverse palindromic recognition sites of length 6. This is because, as we imagine constructing the palindromic sequence, we have four possible options for the first entry, but any of these choices immediately fixes the last entry. This process continues for $n/2$ entries where n is the total length of interest. We thus see that the problem is reduced to finding the possible combinations of half the original string length. The fraction of $2N$-mers, that are reverse palindromic is $\frac{4^N}{4^{2N}} = \frac{1}{4^N}$. We see this as the numerator (before simplification) is the number of possible reverse palindromes in a $2N$-mer, and the denominator (before simplification) is all possible strings for a $2N$-mer.

The following code produces all such possible recognition sites:

```
import itertools

sizeOfPalindrome = 6

#essentially performs a "repeat"-dimensional cartesian product
halfPermutations = itertools.product('ATGC',repeat=sizeOfPalindrome/2)

for permutation in halfPermutations:
        print ''.join(permutation)+''.join(reversed(['A' if b=='T'
        else 'C' if b=='G' else 'T' if b=='A' else 'G' for b in permutation]))
```

2.

The frequencies of the palindromic sequences in the dataset are:

```
{'ACTAGT': 30370, 'TATATA': 144759, 'TCGCGA': 1127, 'TACGTA': 9337,
'CGGCCG': 7651, 'CTTAAG': 50087, 'TGCGCA': 7005, 'CCTAGG': 48979,
 'CTCGAG': 9957, 'GAGCTC': 50049, 'ACATGT': 79458, 'TTCGAA': 8091,
'ATTAAT': 106729, 'AGTACT': 42704, 'AGATCT': 60179, 'ACGCGT': 1631,
'CGTACG': 839, 'GGCGCC': 19486, 'AGCGCT': 8579, 'TGGCCA': 107911,
'ATCGAT': 6471, 'TGTACA': 61405, 'TGATCA': 56893, 'GACGTC': 5317,
'TTTAAA': 252188, 'AAGCTT': 64394, 'GCTAGC': 22197, 'CATATG': 66900,
'CTATAG': 39287, 'TCATGA': 74262, 'AACGTT': 11841, 'CTGCAG': 105663,
'ATGCAT': 66729, 'GCATGC': 43792, 'TAGCTA': 46642, 'AATATT': 170369,
'CGCGCG': 3460, 'CACGTG': 20824, 'CCGCGG': 5530, 'TCCGGA': 7704,
'ATATAT': 161388, 'GATATC': 32152, 'CAATTG': 40935, 'CGATCG': 978,
'ACCGGT': 4216, 'AAATTT': 192451, 'CCCGGG': 31941, 'GCCGGC': 10460,
'GGGCCC': 40294, 'GTGCAC': 38168, 'GTATAC': 31467, 'GTTAAC': 29507,
'TAATTA': 101323, 'GGTACC': 22705, 'GCGCGC': 5048, 'CAGCTG': 89432,
```

```
'GAATTC': 60248, 'TTATAA': 120196, 'GGATCC': 30113, 'TTGCAA': 68694,
'GTCGAC': 2342, 'TCTAGA': 61946, 'AGGCCT': 67067, 'CCATGG': 59851}
```

These frequencies were generated by the following code:

```
with open(sys.argv[1], 'r') as sequenceFile:
        fullSequence = sequenceFile.read()

reversePalindroneFrequencies = {}

for reversePalindrone in reversePalindromes:
        reversePalindroneFrequencies[reversePalindrone] =
        fullSequence.count(reversePalindrone)

print reversePalindroneFrequencies
```

It appears as though the more frequently occurring sequences are those that are made of predominately 'T's and 'A's, while the less frequently occurring recognition sites are those that are made of predominately 'G's and 'C's.

3.

The following code produces a histogram which displays the distribution of lengths of the segments of our sample when cut at recognition sites "CGTAC/G". That is, when the sequence of nucleotides CGTACG are found, a cut is made between the last 'C' and 'G'.

```
import matplotlib.pyplot
import re
cutStrands = re.sub("(?<=CGTAC)(?=G)","jaretrulz",fullSequence).split("jaretrulz")
#super hacky because re.split sucks
matplotlib.pyplot.hist([len(s) for s in cutStrands], bins=100, range=(0,3000000))
matplotlib.pyplot.show()
```

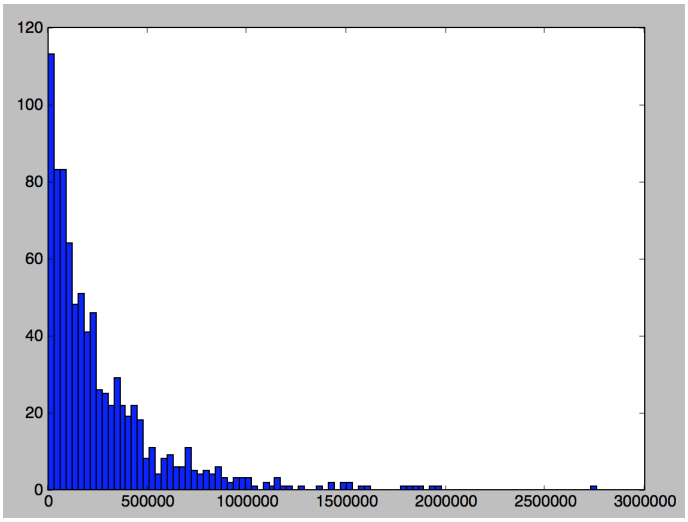

Figure 1: Distribution of DNA fragments by length

4.

```
HapllLengths = [68, 114, 133, 162, 387, 557, 649, 813, 1737, 2012, 2325, 7342]
BstUILengths = [235, 316, 1403, 1455, 1562, 1589, 1633, 1666, 6440]
HapllAndBstUILengths = [9, 68, 99, 114, 133, 136, 162, 183, 387, 513,
 557, 754, 813, 1032, 1455, 1562, 1633, 1666, 2012, 3011]

def powerSet(inputSet):
        powerSet = [[]]
        for element in inputSet:
                powerSet.extend([subset + [element] for subset in powerSet])
        return powerSet

def determineOrderOfSegments(digestionSetA, digestionSetB, combinedDigestionSets):
        powerSetHits = []
        for setToTry in powerSet(combinedDigestionSets):
                if sum(setToTry) in digestionSetA + digestionSetB:
                        powerSetHits.extend([setToTry])

        solutionFound = False

        remainingLengthsToExplain = ["init"]

        while not remainingLengthsToExplain:
                remainingLengthsToExplain = digestionSetA + digestionSetB
                usedPowerSetElements = []
                for powerSetHit in powerSetHits:
                        for powerSetElement in powerSetHit:
                                if powerSetElement in usedPowerSetElements:
                                        break
                        remainingLengthsToExplain.remove(sum(powerSetHit))
                        usedPowerSetElements.extend(powerSetHit)
                print sorted(usedPowerSetElements)

determineOrderOfSegments(HapllLengths, BstUILengths, HapllAndBstUILengths)
```

Collaborators: Alan Wu, Fred Landis.