FREQUENT WORDS WITH MISMATCHES AND REVERSE COMPLEMENT

KOGANTI SRI SAI HARSHITH
AM.EN.U4AIE20139
CSE-AI dept
Amrita School of Engineering
Amritapuri , India
amenu4aie20139@am.students.amrita.edu

Abstract— Biologically, DNA A cannot only bind to perfect DNA A boxes but also bind to their slight variations like Mismatches and reverse complement. Our algorithms will find those mismatches. By using the concepts of Reverse complement and finding frequent words in a pattern algorithm. So, when an input of a DNA sequence and the required hamming distance and the k value for k-mer are given, we can find the DNA A boxes for the given sequence by doing reverse complement of the sequence then find the neighbours. Then the output string from the neighbour method will be passed to the frequent words algorithm which then return the k-mers with the given hamming distance as the output.

Keywords— Reverse Complement, Mis Matches, Hamming Distance, K-mers.

I. INTRODUCTION

DNA replication is a process by which DNA makes a copy of itself. The region where DNA replication begins is known as oriC. DNA A protein is a factor that promotes the unwinding of DNA in the oriC. The initiation phase of DNA replication is determined by the concentration of DNA A protein. Replication begins with active DNA A binding to 9-mer (9-bp) repeats upstream of oriC called DNA Box.
Binding of DNA A to DNA A Box leads to strand separation. DNA A box usually binds between 3 to 15-mer, for bacteria, it is considered as 9-mer. Locating oriC represents an important task for a various biomedical problem

Designing a computational approach to find oriC has mainly two advantages

- Experimental approached to oriC prediction are time-consuming.
- Out of several experiments performed we can locate oriC in a handful number of species.

We have started our study with a Bacterial Genome called Vibrio Cholerae which consists of Two Circular Chromosomes and 4 million base pairs of DNA sequences we have been given the oriC region of Vibrio Cholerae and we found DNA A box (binding sites of DNA A protein)
Which contains a message within the DNA sequence telling the binding site of DNA A using Frequent Words problem and obtained that the 9-mer ATGATCAAG appears three times in the oriC region of Vibrio Cholerae
Some hidden messages are also found using reverse complement the results might be surprising, but DNA A Protein can bind to any of the DNA strands. We applied the Pattern Matching algorithm to identify whether these strings (ATGATCAAG or CTTGATCAT) occur as repeats throughout the genome rather than just in oriC. And finally came to a piece of strong statistical evidence that ATGATCAAG/CTTGATCAT may represent the hidden message to DNA A to start replication.

The Escherichia coli (E. Coli) genome consists of circular DNA and 46,00,000 base pairs. Solving the Minimum Skew Problem provided an approximate location of oriC at position 3923620 in E. coli.
From the case of Vibrio Cholerae, it is observed that a DNA A box may appear with slight variations like mismatches and reverse compliments which can be solved by using Frequent Words with Mismatches and Reverse Complements Problem which help us to find DNA A Boxes in E. Coli The experimentally confirmed DNA A box in E. coli (TTATCCACA) is a most frequent 9-mer with 1 mismatch, along with its reverse complement TGTGGATAA which matched with our output

II. LITERATURE REVIEW

Bioinformatics is an extremely encouraging field that has such countless applications in the aspects of agribusiness, microbiology, biomedicine etc. (Hogweed and Hesper, 1970; Hogeweg,1978, 2011) made up the term bioinformatics as the investigation of data measures in biotic frameworks. This definition set bioinformatics in equal connection to biophysics which manages the investigation of actual cycles in natural frameworks or biochemistry. Computers became significant in atomic science when protein arrangements became available after Frederick Sanger decided the grouping of insulin in the mid-1950s.Comparing numerous successions physically ended up being unreasonable. Margaret Oakley Dalhoff, a pioneer in bioinformatics, who has been hailed by David Lipman, director of the National Centre for Biotechnology Information, as the "mother and father of bioinformatics, (Moody, 2004), arranged one of the first protein sequence databases, at first distributed as books and spearheaded strategies for succession alignment and sub-atomic development.

III. METHODOLOGY

For finding DNA A boxes that bind to their slight variations like Mismatches and reverse complements, we redefine the Frequent Words Problem's algorithm to account for both mismatches and reverse complements.

Algorithm:

```
FrequentPatterns ← an empty set
Neighborhoods ← an empty list
for i = 0 to |Text| − k do
        add NEIGHBORS(Text(i,k), d) to Neighborhoods
        add NEIGHBORS(Text(i, k), d) to Neighborhoods
form an array NeighborhoodArray holding all strings in Neighborhoods
for i = 0 to |Neighborhoods| − 1 do
        Pattern ← NeighborhoodArray(i)
        Index(i) ← PATTERNTONUMBER(pattern)
        Count(i) ← 1
SortedIndex ← SORT(Index )
for i = 0 to |Neighborhoods| − 2 do
        if SortedIndex(i) = SortedIndex(i+1) then
                Count(i+1) ← Count(i)+1
maxCount ← maximum value in array Count

for i = 0 to |Neighborhoods| − 1 do
        if Count(i) = maxCount then
                Pattern ← NUMBERTOPATTERN(SortedIndex(i), k)
                add Pattern to FrequentPatterns
return FrequentPatterns
```

Hamming Distance:

We say that position i in k-mers p1 · · · pk and q1 · · · qk is a mismatch if pi != qi. The number of mismatches between strings p and q is called the Hamming distance between these strings and is denoted HAMMINGDISTANCE(p, q).

Hamming Distance:

```
Count ← 0
for i = 0 to |a| do
if a[i] != b[i] then
        Count ← Count + 1
return Count
```

Reverse Complement:

Given a nucleotide p, we denote its complementary nucleotide as p. The reverse complement of a string Pattern = p1 ··· pn is the string Pattern = pn ··· p1 formed by taking the complement of each nucleotide in Pattern, then reversing the resulting string.

Reverse Complement:

```
rText ← reverse
Text ← an empty string
for i = 0 to |rText| − 1 do
Text(i) ← complement of rText(i)
return Text
```

NEIGHBORS:

Given a k-mer Pattern, we therefore define its d-neighborhood NEIGHBORS(Pattern, d) as the set of all k-mers that are close to Pattern.

NEIGHBORS (Pattern, d):

```
if d = 0 then
return {Pattern}
if |Pattern| = 1 then
return {A,C,G,T}
Neighborhood ← an empty set
SuffixNeighbors ← NEIGHBORS(Suffix(Pattern), d)
for each string Text from SuffixNeighbors do
  if HAMMINGDISTANCE(Suffix(Pattern), Text) < d then
```

```
        for each nuceotide x do add x + Text to
                            Neighborhood
        else
        add FirstSymbol(Pattern) + Text to
                            Neighborhood
    return Neighborhood
```

NUMBERTOPATTERN:

In order to compute the inverse function NUMBERTOPATTERN(index, k), we return to above, which implies that when we divide index = PATTERNTONUMBER(Pattern) by 4, the remainder will be equal to SYMBOLTONUMBER(symbol), and the quotient will be equal to PATTERNTONUMBER(PREFIX(Pattern)). Thus, we can use this fact to peel away symbols at the end of Pattern one at a time.

NUMBERTOPATTERN(index, k):

```
    if k = 1 then
        return
    NUMBERTOSYMBOL(index)
    prefixIndex ← QUOTIENT(index, 4)
    r ← REMAINDER(index, 4)
    PrefixPattern ←
    NUMBERTOPATTERN(prefixIndex, k-1)
    return concatenation of PrefixPattern with
    symbol
```

PATTERNTONUMBER:

Our approach to computing PATTERNTONUMBER(Pattern) is based on a simple observation. If we remove the final symbol from all lexicographically ordered k-mers, the resulting list is still ordered lexicographically (think about removing the final letter from every word in a dictionary). In the case of DNA strings, every (k-1) mers in the resulting list is repeated four times.

PATTERNTONUMBER(Pattern):

```
    if Pattern contains no symbols then
        return 0
    symbol ← LastSymbol(Pattern)
    Prefix ← PREFIX(Pattern)
    return  PATTERNTONUMBER(Prefix ) +
    SYMBOLTONUMBER(symbol)
```

IV. RESULT

Input:

```
CTTGCCGGCGCCGATTATACGATCGCGGCCGCTTGCCTTC
TTTATAATGCATCGGCGCCGCGATCTTGCTATATACGTAC
GCTTCGCTTGCATCTTGCGCGCATTACGTACTTATCGATT
ACTTATCTTCGATGCCGGCCGGCATATGCCGCTTTAGCAT
CGATCGATCGTACTTTACGCGTATAGCCGCTTCGCTTGCC
GTACGCGATGCTAGCATATGCTAGCGCTAATTACTTAT
9 3
```

Output:

```
AGCGCCGCT AGCGGCGCT
```

Input:

```
ACGTTGCATGTCGCATGATGCATGAGAGCT
4 1
```

Output:

```
ACAT ATGT
```

V. CONCLUSION

By using the improvised version of the frequent words problem i.e. The Frequent words with mismatches and reverse complement problem, we can find DNA A boxes in the oriC region of the DNA sequence. The experimentally obtained DNA A boxes are the most frequent k-mer with d mismatch, along with its reverse complement.

VI. APPENDIX

```
from collections import defaultdict

def HammingDistance(seq1,seq2):
   d=0
   for i in range(len(seq1)):
      if seq1[i]!=seq2[i]:
         d+=1;
   return d

def ReversePattern(pattern):
   temp=pattern.replace("A","X").replace("T","A").rep
lace("X","T")
   return
(temp.replace("G","X").replace("C","G").replace("X",
"C")[::-1])

def neighbour(pattern, mismatch, words):
   if mismatch == 0:
      words.add(pattern)
   else:
      bases = ['A', 'T', 'C', 'G']
      for i in range(len(pattern)):
         for j in range(len(bases)):
            new_pattern = pattern[:i] + bases[j] +
pattern[i+1:]
            if mismatch <= 1:
               words.add(new_pattern)
            else:
               neighbour(new_pattern, mismatch-1,
words)

def FindMostFrequentPattern(text, k, d):
   allfrequentwords = defaultdict(int)
```

```
    for i in range(len(text) - k + 1):
        frequentwords = set()
        neighbour(text[i:i + k], d, frequentwords)

        for words in frequentwords:
            allfrequentwords[words] += 1

    for t in allfrequentwords.keys():
        reverse_k = ReversePattern(t)
        for i in range(len(text) - k + 1):
            if HammingDistance(text[i:i + k], reverse_k)
<= d:
                allfrequentwords[t] += 1

    result = set()
    for t in allfrequentwords.keys():
        if allfrequentwords[t] ==
max(allfrequentwords.values()):
            result.add(t)
            result.add(ReversePattern(t))
    for i in result:
        print(i, end=" ")

in_file = open('rosalind_ba1j.txt', 'r')
text = in_file.readline().strip()
k, d = map(int, in_file.readline().strip().split())
FindMostFrequentPattern(text, k, d)
```

References

[1] https://amritauniv.sharepoint.com/:p:/s/19BIO112IntelligenceofBiologicalSystems2-S2AIEB/EZTxdYlq839Mopap6QZA_osBsi1Uq5tgqf_V15XKfnmo0Q?e=AKnyHF

[2] http://rosalind.info/problems/ba1j/

[3] https://bioinformaticsalgorithms.com/data/debugdatasets/replication/FrequentWordsWithMismatchesAndReverseComplementsProblem.pdf

[4] https://amritauniv.sharepoint.com/:p:/s/19BIO112IntelligenceofBiologicalSystems2-S2AIEB/EUy5udhfCm5Hlmcvc8SyF8cB1EL0FGsR5nHVxTReigSbVw?e=gnArYK

[5] Compeau, Phillip, and Pavel Pevzner. Bioinformatics algorithms: an active learning approach. Active Learning Publishers, 2015