Improved algorithm for frequent words problem : frequent words with mismatches by sorting

21BIO112 Intelligence of biological systems

Group-13

ADITYA VARDHAN REDDY-AM.EN.U4AIE21171

ASHWIN ARJUNAN-AM.EN.U4AIE21118

SHREYA PULLURI-AM.EN.U4AIE21159

MAHADEV-AM.EN.U4AIE21141

SATVIK CHOULAPALLY-AM.EN.U4AIE21157

***Abstract*—In the genome of every species there exists a replication origin known as oric where the replication of DNA starts. Finding the replication origin oric is very important in any bioinformatics research. In this project we are going to built a improved algorithm for frequent words with mismatches by sorting. We are using python as our programming language for solving the problem. We are finding a solution for finding replication origin (oric)**

Keywords— Replication origin oric, DNA replication

I. INTRODUCTION:

An important property of DNA is that it can replicate, or make copies of itself. Each strand of DNA in the double helix can serve as a pattern for duplicating the sequence of bases. This is critical when cells divide because each new cell needs to have an exact copy of the DNA present in the old cell.

(A) DNA:

DNA, or deoxyribonucleic acid is the hereditary material in human and almost all other organism. Nearly every cell in a person's body has the same DNA. The information in DNA is stored as a code made up of four chemical bases: adenine (A), guanine (G), cytosine (C), and thymine(T). Human DNA consists of about 3 billion bases and more than 99 percent of those bases are the same in all people. The order or sequence of these bases determines the information available for building and maintaining an organism. DNA bases pair up with each other, A with T and C with G, to form units called base pairs.

(B) REPLICATION:

DNA replication is the process by which the genome's DNA is copied in cells. Before a cell divides, it must first copy its entire genome so that each resulting daughter cell ends up with its own complete genome. Replication begins at a location on the double helix known as oric. An enzyme known as DNA helicase disrupts the hydrogen bonding. The two strands are separated into single stranded Y shape structure known as the replication fork. This area will be the template for replication to begin

(C) DNA A AND DNA A BOX

DnaA is a protein that activates initiation of DNA replication in bacteria. It is a replication initiation factor which promotes the unwinding of DNA at Oric. The onset of the initiation phase of DNA replication is determined by the concentration of DnaA. Replication begins with active DnaA binding to 9-mer(9-bo) repeats upstream of

oric called DnaA Box. Binding of DnaA to DnaA box leads to strand separation.

(D) PROBLEM STATEMENT:

By using advanced methods like the minimum skew problem, we find an approximate location of Oric at position 3923620 in E. coli Genome. After using the Frequent Words Problem to solve in a window of length 500 starting at position 3923620 of the genome reveals no 9-mers that appear three or more time. Even if we have managed to locate Oric in E. coli, it appears that we still have not found the DnaA boxes that jump-start replication in this bacterium.

(E) OBJECTIVE:

In this project our goal is to modify our previous algorithm for the frequent words with mismatches by sorting in order to find DnaA boxes. By identifying frequent k-mers with mismatches by sorting. A most frequent k-mer with up to d mismatches in Text is simply a string Pattern maximizing count (Text, pattern) among all K-mers. Note that pattern does not need to actually appear as a substring of Text, for example, AAAAA is the most frequent 5-mer with 1 mismatch in AACAAGCTGATAAACATTTAAAGAG, even though AAAAA does not appear exactly in this string.

II. METHODOLOGY:

A. Hamming distance :

The Hamming distance between two equal-length strings of symbols is the number of positions at which the corresponding symbols are different. The hamming distance also gives the number of mismatches between two DNA strings of a given length. It is one of the essential components of the algorithm which we are going to develop for frequent words with mismatches by sorting. The algorithm for finding out the hamming distance between to Strings is given below-:

• Iterate through each character in the first string

• While iterating, check if the character at any given point in the first string is equal to the character, if not it corresponds to a mismatch

• Find out the total number of mismatches

The below Code is used in python for hamming distance problem:

```
def hamming_distance(patt1 , patt2):

    hd = 0

    if(len(patt1)!=len(patt2)):

        print("Error!! length should be same for finding hamming distance")

        return
    else:
        for i in range(len(patt1)):

            if(patt1[i]!=patt2[i]):

                hd += 1

    return hd
```

B. D- Neighbourhood :

The d-neighborhood of the k-mer pattern is the collection of all k-mers that are at most Hamming distance d from pattern

The below code is used in python for d-neighbourhood

```
def d_neighbourhood(text,d):

    neighbours = []

    lst = generate_unique_pattern(text)

    for i in lst:

        if(hamming_distance(i,text)<=d):

            neighbours.append(i)

    return neighbours
def generate_unique_pattern(pattern):

    lst = list(itertools.product("ACGT",repeat=len(pattern)))

    for i in range(len(lst)):

        str = ""

        for j in lst[i]:

            str+=j

        lst[i]=str

    return lst
```

C. Pattern to number :

Our approach to computing PATTERN TO NUMBER (pattern) is based on simple observation. If we remove the final symbol from all lexicographically ordered k-mers, the resulting list is still ordered lexicographically. In the case of DNA strings, every (k-1)-mer in the resulting list is repeated four times.

AAA	AAC	AAG	AAT	ACA	ACC	ACG	ACT
AGA	AGC	AGG	AGT	ATA	ATC	ATG	ATT
CAA	CAC	CAG	CAT	CCA	CCC	CCG	CCT
CGA	CGC	CGG	CGT	CTA	CTC	CTG	CTT
GAA	GAC	GAG	GAT	GCA	GCC	GCG	GCT
GGA	GGC	GGG	GGT	GTA	GTC	GTG	GTT
TAA	TAC	TAG	TAT	TCA	TCC	TCG	TCT
TGA	TGC	TGG	TGT	TTA	TTC	TTG	TTT

Thus, the number of 3-mers occurring before AGT is equal to four times the number of 2-mers occurring before AG plus the number of 1-mers occurring before T. Therefore,

Patterntonumber(AGT) = 4.patterntonumber(AG) + Symboltonumber(T)

The below python code is used for pattern to number

```
def patt_to_num(patt):
    if patt=="":
        return 0
    symbol = LastSymbol(patt)
    prefixpatt = prefix(patt)

    return 4*patt_to_num(prefixpatt) + Symbol_to_num(symbol)
```

D. Number to pattern :

In order to compute the inverse function NUMBER TO PATTERN (index, k), we divide index = PATTERN TO NUMBER (pattern) by 4, the reminder will be equal to SYMBOLTONUMBER (symbol), and the quotient will be equal to PATTERNTONUMBER (PREFIX (pattern)). we can use this fact to remove symbols at the end of pattern one at a time.

The below python code is used for number to pattern

```
def num_to_patt(index,k):
    if k ==1:
        return num_to_symbol(index)
    prefixind = index//4
    r= index % 4
    symbol = num_to_symbol(r)
    prefixpat = num_to_patt(prefixind,k-1)
    return prefixpat+symbol
```

E. Symbol to number :

SYMBOLTONUMBER(symbol) is the function transforming symbols A,C,G, and T into the respective integers 0,1,2, and 3.

The below python code is used for symbol to pattern

```
def Symbol_to_num(patt):
    symbol_map = {"A":0,"C":1,"G":2,"T":3}
    temp = 0
    for i in patt:
        temp = symbol_map.get(i)
    return temp
```

F. Number to symbol:

NUMBERTOSYMBOL is function transforming Numbers 0,1,2, and 3 into their respective symbols A,C,G, and T.

The below python code is used for Number to symbol

```
def num_to_symbol(patt):
    symbol_map = {0:"A",1:"C",2:"G",3:"T"}

    if symbol_map.get(patt):
        return symbol_map.get(patt)
    else:
        return ""
```

G. Last symbol and prefix pattern :

If we remove the final symbol of pattern, denoted LASTSYMBOL(Pattern), then we will obtain a (k-1)- mer that we define as PREFIX(Pattern). The observation therefore generalizes to the pattern to number formula.

The below python code is used for last symbol

```
def LastSymbol(patt):
    try:
        return patt[-1]
    except Exception:
        pass
```

The below python code is used for prefix pattern

```
def prefix(patt):

    return patt[0:-1]
```

H. Frequentwords with mismatches by sorting:

Given a string text whose k = 2 , list all its 2-mers in the order they appear in Text, and convert each 2-mer into an integer using PATTERNTONUMBER to produce an array INDEX. We will now sort INDEX to generate an array SORTEDINDEX. Since identical k-mers clump together in the sorted array, frequent k-mers are the longest runs of identical integers in Sorted Index. This insights leads to Frequent words with mismatches by sorting

The below is the python code for FREQUENT WORDS WITH MISMATCHES BY SORTING

```
def findfreqwordswithmismatchbysort(text,k,d):

    freq_patterns = []

    neighbourhoods = []

    index = []

    count = []

    for i in range(len(text)-k):

        neighbourhoods.extend(d_neighbourhood(text[i:i+k],d))

    for i in range(len(neighbourhoods)-1):

        pattern = neighbourhoods[i]

        index.append(patt_to_num(pattern))

        count.append(1)

    temp = []


    for i in index:

        if(i!=None):

            temp.append(int(i))

    index=temp

    index.sort()

    for i in range(len(index)-1):

        if index[i] == index[i+1]:

            count[i+1]=count[i]+1

maxcount = max(count)

for i in range(len(neighbourhoods)-1):

    if(count[i]==maxcount):

        pattern = num_to_patt(index[i],k)

        freq_patterns.append(pattern)

return freq_patterns
```

III.ALGORITHMS:

FINDINGFREQUENTWORDSWITHMISMATCHESBYSORTING(*Text*, *k*, *d*)
  FrequentPatterns ← an empty set
  Neighborhoods ← an empty list
  for *i* ← 0 to |*Text*| − *k*
    add NEIGHBORS(*Text*(*i*, *k*), *d*) to *Neighborhoods*
  form an array NEIGHBORHOODARRAY holding all strings in *Neighborhoods*
  for *i* ← 0 to |*Neighborhoods*| − 1
    Pattern ← NEIGHBORHOODARRAY(*i*)
    INDEX(*i*) ← PATTERNTONUMBER(*Pattern*)
    COUNT(*i*) ← 1
  SORTEDINDEX ← SORT(INDEX)
  for *i* ← 0 to |*Neighborhoods*| − 1
    if SORTEDINDEX(*i*) = SORTEDINDEX(*i* + 1)
      COUNT(*i* + 1) ← COUNT(*i*) + 1
  maxCount ← maximum value in array COUNT
  for *i* ← 0 to |*Neighborhoods*| − 1
    if COUNT(*i*) = *maxCount*
      Pattern ← NUMBERTOPATTERN(SORTEDINDEX(*i*), *k*)
      add *Pattern* to *FrequentPatterns*
  return *FrequentPatterns*

PATTERN TO NUMBER:

PATTERNTONUMBER(*Pattern*)
  if *Pattern* contains no symbols
    return 0
  symbol ← LASTSYMBOL(*Pattern*)
  Prefix ← PREFIX(*Pattern*)
  return 4 · PATTERNTONUMBER(*Prefix*) + SYMBOLTONUMBER(*symbol*)

NUMBER TO PATTERN:

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

IV. RESULTS:

I. DATASET:

Datasets are taken from https://rosalind.info/problems/ba1i/

1) DNA-

```
ACGTTGCATGTCGCATGATGCATGAGAGCT

K value = 4  d value = 1
```

OUTPUT:

ATGC ATGT GATG

2) DNA-

GTATTTTATTGTTGTTTTGTTGTTCTCAGTCAATTTT
GCAGTATTTTATTGTTGTTTTGTTGTTCTCAGTCACT
CAGTCAGTATTTTACTCAGTCAGTATTTTAATTTTG
CACGGCTTTACTTGTTGTTGTATTTTAGTATTTTAAT

TTTGCAGTATTTTACGGCTTTACCTCAGTCATTGTT
GTTCGGCTTTACCTCAGTCAATTTTGCAATTTTGCA
GTATTTTATTGTTGTTCTCAGTCAATTTTGCAGTATT
TTACGGCTTTACCGGCTTTACTTGTTGTTATTTTGC
ACTCAGTCATTGTTGTTCTCAGTCACTCAGTCATTG
TTGTTCTCAGTCAGTATTTTATTGTTGTTTTGTTGTT
GTATTTTAATTTTGCAGTATTTTACGGCTTTACCTC
AGTCAGTATTTTACTCAGTCATTGTTGTTCTCAGTC
ATTGTTGTTATTTTGCATTGTTGTTGTATTTTACGGC
TTTACCTCAGTCATTGTTGTTATTTTGCATTGTTGTT
ATTTTGCACTCAGTCACTCAGTCAGTATTTTATTGT
TGTTATTTTGCACTCAGTCAGTATTTTAATTTTGCA
CGGCTTTACCTCAGTCACTCAGTCACGGCTTTACCT
CAGTCACTCAGTCACGGCTTTACCGGCTTTACCGGC
TTTACATTTTGCACTCAGTCACGGCTTTACCTCAGT
CAATTTTGCACTCAGTCAGTATTTTACGGCTTTACC
GGCTTTACATTTTGCACGGCTTTACCTCAGTCAGTA
TTTTATTGTTGTTTTGTTGTTCTCAGTCACGGCTTTA
CCGGCTTTACCTCAGTCAATTTTGCAGTATTTTAAT
TTTGCATTGTTGTTGTATTTTAGTATTTTAGTATTTT
ACGGCTTTACCTCAGTCAGTATTTTACTCAGTCACT
CAGTCACTCAGTCA

K value – 7 d value - 2

OUTPUT:
TTTTTTT

3) DNA-

CATAAGCACTCTTGGACTCTTGGGATCACTTCGATC
ACTTCCATAAGCGATCACTTCTCGAAGACATAAGC
ACTCTTGGTCGAAGACATAAGCCATAAGCTCGAAG
AACTCTTGGATTCAAGGTCGAAGAATTCAAGGCAT
AAGCCATAAGCCATAAGCTCGAAGACATAAGCTCG
AAGAATTCAAGGGATCACTTCACTCTTGGACTCTT
GGCATAAGCATTCAAGGATTCAAGGATTCAAGGTC
GAAGACATAAGCTCGAAGAACTCTTGGTCGAAGAA
CTCTTGGATTCAAGGTCGAAGAATTCAAGGATTCA
AGGTCGAAGACATAAGCACTCTTGGTCGAAGAACT
CTTGGCATAAGCATTCAAGGCATAAGCATTCAAGG
ATTCAAGGACTCTTGGGATCACTTCTCGAAGAGAT
CACTTCTCGAAGAACTCTTGGCATAAGCATTCAAG
GCATAAGCCATAAGCATTCAAGGGATCACTTCATT
CAAGGCATAAGCATTCAAGGACTCTTGGCATAAGC
ATTCAAGGTCGAAGAGATCACTTCATTCAAGGGAT
CACTTCCATAAGCTCGAAGAACTCTTGGACTCTTG
GTCGAAGAATTCAAGGATTCAAGGACTCTTGGCAT
AAGCATTCAAGGACTCTTGGCATAAGCACTCTTGG
TCGAAGAGATCACTTCACTCTTGGGATCACTTCTCG
AAGACATAAGCACTCTTGGTCGAAGAGATCACTTC
TCGAAGATCGAAGAACTCTTGGCATAAGCATTCAA
GGTCGAAGACATAAGCACTCTTGGCATAAGCTCGA
AGATCGAAGAACTCTTGGGATCACTTC

K-Value = 7 d-Value = 2

OUTPUT:
TCTAGGA

V.CONCLUSIONS:

From this project we can conclude that frequent words with mismatches by sorting is an faster algorithm for finding solution for frequent words with mismatches problem. The dataset taken from rosaline has given correct output for every given dataset. Another thing which we can conclude is that bioinformatics when combined with computational engineering techniques can help us to identify the basics of life.

VI. ACKNOWLEDGMENT:

We the members of group 13 would like to show our gratitude towards Dr. Manjusha Nair, for helping us in the field of bioinformatics and putting in a lot of effort for explanation of the experimental approach in the field of Bioinformatics. We would also like to thank all the other faculty who helped us through out the semester in the field of bioinformatics.

VII. REFERENCES:

- https://rosalind.info/problems/ba1i/
- https://amritauniv.sharepoint.com/sites/19BIO112IntelligenceofBiologicalSystems2-S2AIEB/Shared%20Documents/Forms/AllItems.aspx?id=%2Fsites%2F19BIO112IntelligenceofBiologicalSystems2%2DS2AIEB%2FShared%20Documents%2FGeneral%2FBooks%2F2%5FTextBook%5FBioinformaticsAlgorithms%2Epdf&parent=%2Fsites%2F19BIO112IntelligenceofBiologicalSystems2%2DS2AIEB%2FShared%20Documents%2FGeneral%2FBooks&p=true&ga=1