Frequent k-mers in the SARS-CoV-2 Genome

CS516 Bioinformatics - Project 1

Israel Gonzalez S., Gabriel Romero

2/22/23

Table of contents

1 Abstract

Write here the abstract (Gabriel)

2 Introduction

Write here the intro (Gabriel)

3 Methods

We proceed in this section to describe the methods we have implemented in the Python class FrequentKmers and the way they can be used in an executable Python file main.py

3.1 Code

The methods are organized in the way each task required. First, we have the methods related to Tasks 1 and Task 2 that require to code the pseudo code we have in *Bioinformatics Algorithms– An Active Learning Approach, 3rd Edition*.

The class encapsulates all methods is:

```
class FrequentKmers:
```

```
  # Frequent Words Algorithm methods:
  def frequent_words(self, text, k):
    start = timer()
    frequent_words = []
    n = len(text)
    count_array = np.zeros(n - k + 1)
    for i in range(n - k + 1):
      pattern = text[i:i+k]
      count_array[i] = self.pattern_count(text, pattern)
```

```
    for i in self.max_array(count_array):
      frequent_words.append(text[i:i+k])
    frequent_words = self.remove_duplicates(frequent_words)
    end = timer()
    return [frequent_words,  round((end - start) * 1000, 3)]

  def pattern_count(self, text, pattern):
    count = 0
    for i in range(len(text) - len(pattern) + 1):
      if text[i:i+len(pattern)] == pattern:
        count += 1
    return count

  def remove_duplicates(self, arr):
    answer = []
    for item in arr:
      if item not in answer:
        answer.append(item)
    return answer

  def max_array(self, arr):
    max_val = max(arr)
    max_indices = [i for i, val in enumerate(arr) if val == max_val]
    return max_indices
```

The `FrequentKmers.frequent_words` method takes as parameters `text` and `k` representing the DNA sequence and the length of k-mer respectively. Taking text slices of length k, we *count* the frequency of each k-mer using `FrequentKmers.pattern_count` method. After counting them, we determine the k-mers with a maximum frequency count using `FrequentKmers.max_array` and filter them in a unique list without repetitions using `FrequentKmers.remove_duplicates`

```
  # Better Frequent Words Algorithm methods:
  def better_frequent_words(self, text, k):
    start = timer()
    freq_map = self.frequent_map(text, k)
    if freq_map is not None:
      max_count = max(freq_map.values())
      frequent_words = []
      for pattern in freq_map.items():
        if pattern[1] == max_count:
```

```
            frequent_words.append(pattern[0])
        end = timer()
        return [frequent_words, round((end - start) * 1000, 3)]
    else:
        end = timer()
        return[None, round((end - start) * 1000, 3)]

def frequent_map(self, text, k):
    freq_map = {}
    n = len(text)
    for i in range(n - k + 1):
        pattern = text[i:i+k]
        if pattern in freq_map:
            freq_map[pattern] += 1
        else:
            freq_map[pattern] = 1
    return freq_map
```

Similarly, the `FrequentKmers.better_frequent_words` method takes as parameters `text` and `k` representing the DNA sequence and the length of k-mer respectively.

Explaination (Israel)

3.2 Execution

Explaination (Israel)

4 Results

In this section, we proceed to describe our results. We have structured this section in the same schema the project tasks appears in the project prompt.

4.1 Algorithms for finding frequent k-mers

4.1.1 Task 1. Frequent words by counting

This is the output of our program:

```
==========================================
Task 1: Algorithm Frequent Words (counting)
==========================================
Using test cases from class slides (Ch 1 slides 64 and 99),
we show the list of frequent words and running time in ms for our implementation
are correct:
[['ACG', 'TTT'], 0.163]
[['atgatcaag', 'ctcttgatc', 'tcttgatca', 'cttgatcat'], 89.532]
```

Comment these results here (Gabriel)

4.1.2 Task 2. Frequent words by hashing

This is the output of our program:

```
==================================================
Task 2: Algorithm Better Frequent Words (hashing)
==================================================
Using test cases from class slides (Ch 1 slides 64 and 99),
we show the list of frequent words and running time in ms for our implementation
are correct:
[['ACG', 'TTT'], 0.1]
[['atgatcaag', 'ctcttgatc', 'tcttgatca', 'cttgatcat'], 0.792]
```

Comment these results here (Gabriel)

4.2 Understanding the runtime

4.2.1 Task 3. Method generates random DNA sequences of several random lengths

This is the output of our program:

```
==========================================
Task 3: Generating Random DNA Sequences
==========================================
Using Numpy random choice function of Python,
we generate 10 random DNA sequences of variable lenght:
'GACTCGGATATGT'
'GAATATGTGCAA'
'AGGGGATACTACT'
```

‘GTACATATCA’
‘TTGGGATAACA’
‘GTGGGAAGCACGT’
‘ACATACAGACGCA’
‘ATTCTTAACAAGGC’
‘CTATTGAAAG’
‘TCCGTAGGTCT’

Comment these results here (Gabriel)

4.2.2 Task 4. Analysis of empirical time comparing counting and hashing algorithms

To empirically compare the two algorithms in terms of runtime, our program for this analysis has produced the comparison in the following charts, and also consolidated everything in a major table with these results.

The following are our charts:

And in a consolidated table:

Length	K-length	FW(ms)	BFW(ms)
25	3	0.369	0.04
25	6	0.263	0.029
25	9	0.201	0.026
25	12	0.148	0.022
25	15	0.117	0.019
50	3	2.672	0.107
50	6	2.848	0.149
50	9	2.291	0.121
50	12	1.878	0.132
50	15	1.687	0.112
75	3	7.455	0.151
75	6	7.312	0.224
75	9	5.965	0.155
75	12	6.064	0.214
75	15	6.115	0.143
100	3	9.436	0.21
100	6	10.619	0.19
100	9	9.054	0.316
100	12	7.918	0.24
100	15	6.889	0.117
250	3	55.049	0.269

Length	K-length	FW(ms)	BFW(ms)
250	6	35.782	0.282
250	9	42.438	0.283
250	12	26.598	0.276
250	15	27.295	0.288
500	3	198.86	0.467
500	6	125.661	0.563
500	9	156.53	0.874
500	12	157.304	0.867
500	15	157.182	2.094
750	3	334.105	0.696
750	6	261.337	0.798
750	9	291.711	0.824
750	12	259.805	0.803
750	15	268.239	0.808
1000	3	602.367	0.915
1000	6	501.437	1.017
1000	9	475.136	0.967
1000	12	476.967	1.067
1000	15	472.738	1.058

Comment these results here (Gabriel)

4.3 Frequent k-mers in the SARS-Cov-2 genome

4.3.1 Task 5 and Task E2. SARS-CoV-2 and SARS-CoV most frequent k-mers

The following is a table of the most frequent k-mers present in the SARS-CoV-2 and SARS-CoV genomes:

Genome	K-length	K-mer
SARS-CoV-2	3	TTT
SARS-CoV-2	6	TTGTTA
SARS-CoV-2	9	TAAACGAAC
SARS-CoV-2	12	GTTGATGGTGTT
SARS-CoV-2	15	ATCAGACAACTACTA
SARS-CoV-2	15	TCAGACAACTACTAT
SARS-CoV-2	15	CAGACAACTACTATT
SARS-CoV-2	15	CAATTATTATAAGAA
SARS-CoV-2	15	AATTATTATAAGAAA

Genome	K-length	K-mer
SARS-CoV-2	15	ATTATTATAAGAAAG
SARS-CoV-2	15	TTGCAGAGTGGTTTT
SARS-CoV-2	15	AAAGTTGATGGTGTT
SARS-CoV-2	15	AAGTTGATGGTGTTG
SARS-CoV-2	15	TAAACGAACATGAAA
SARS-CoV	3	TTT
SARS-CoV	6	TTGCTG
SARS-CoV	9	TAAACGAAC
SARS-CoV	12	TGAGGAAGAAGA
SARS-CoV	12	TAAAATGTCTGA
SARS-CoV	15	AAAAGAAAAAGACTG
SARS-CoV	15	AAAGAAAAAGACTGA
SARS-CoV	15	ATTATAATTATAAAT

Comment these results here (Gabriel)

4.3.2 Task 6. Interpreting the k-mers found in a biological context

Comment your research about this question here (Gabriel)

4.4 Extra Credits

4.4.1 Task E1. The null distribution of the frequency of most frequent k-mers

Our program produced these results:

```
==========================================================
E1: Null Distribution of the most frequent k-mers in SARS-CoV-2 genome sequence
==========================================================
Using the SARS-CoV-2 genome sequence, our permutation test
with 1000 permutations gives. We have considered a p-value threshold of 0.05 to
decide
if the most frequent k-mer is unlikely to have occurred by chance alone
and consequently, it has biological significance:


3-mers give a p-value of 0.33067, then they DO NOT HAVE biological significance
6-mers give a p-value of 0.34366, then they DO NOT HAVE biological significance
9-mers give a p-value of 0.02198, then they DO HAVE biological significance
```

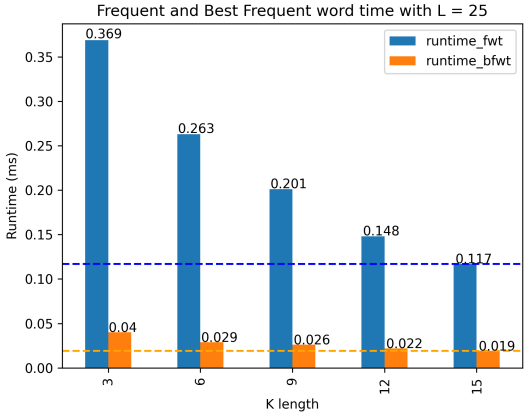


Figure 1: L25

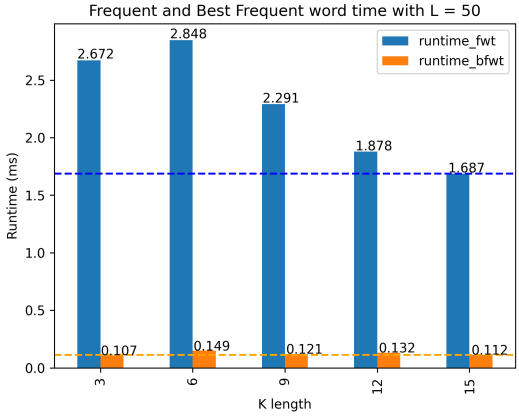


Figure 2: L50

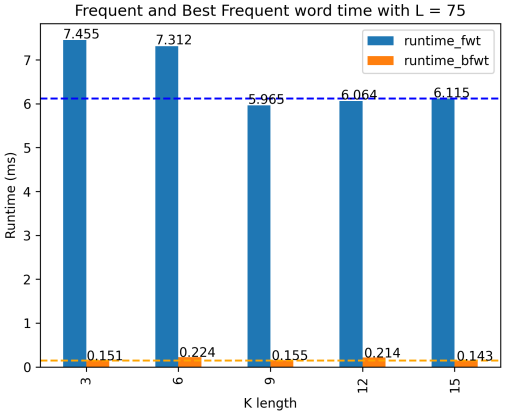


Figure 3: L75

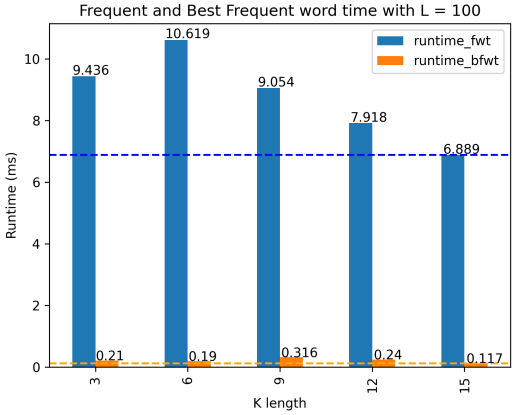


Figure 4: L100

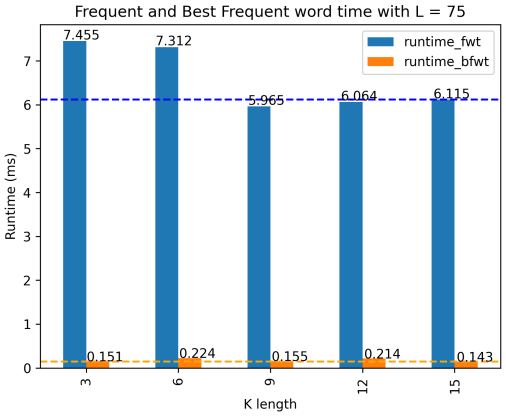


Figure 5: L250

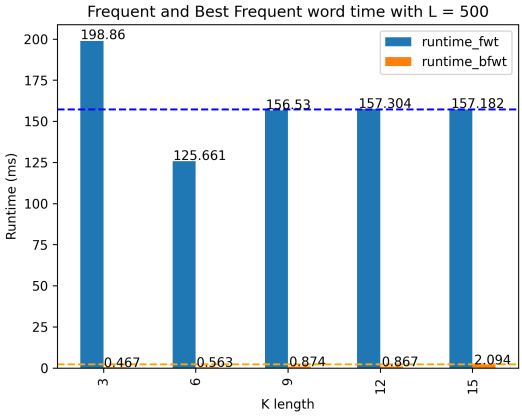
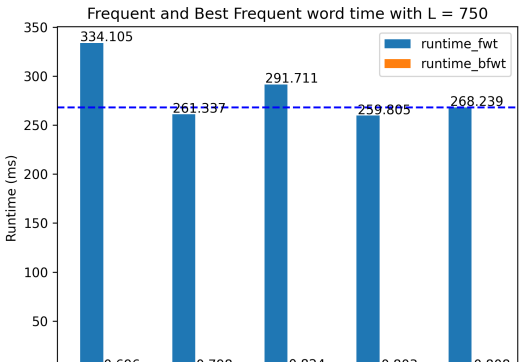
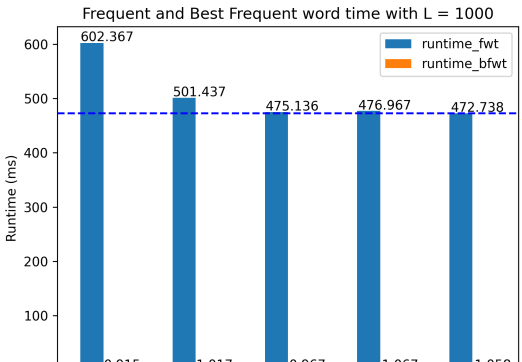


Figure 6: L500





12-mers give a p-value of 0.09491, then they DO NOT HAVE biological significance
15-mers give a p-value of 0.52248, then they DO NOT HAVE biological significance

Comment your research about this question here (Gabriel)

4.4.2 Task E2. SARS-CoV versus SARS-CoV-2

Analyzed above in Task 5. Please, go to that section.

4.4.3 Task E3. Find the longest k-mer repeats beyond 15

First, we find what is the max value for k in each Genome:

Genome	K-length	# matches
SARS-CoV-2	15	10
SARS-CoV-2	16	5
SARS-CoV-2	17	2
SARS-CoV	15	3
SARS-CoV	16	1

Now, that we know for SARS-CoV-2 is k=17 and for SARS-CoV is K=16, we found the longest kmers: Longest k-mers in SARS-CoV-2: ['ATCAGACAACTACTATT', 'CAATTATTATAAGAAAG'] Longest k-mers in SARS-CoV : ['AAAAGAAAAAGACTGA']

Comment your research about this question here (Gabriel)

4.4.4 Task E4. Develop a fast algorithm to find k-mer repeats in a given range of k

Pending of coding. Part of future work.

4.4.5 Task E5. Evolution of repeats in coronavirus genomes

Pending of coding. Part of future work.

5 Discussion

Write here (Gabriel)

6 Distribution of work

Write here (Gabriel)

7 References

Write here (Gabriel)