Comparative analysis of algorithms to find frequent k-mers and their implementation in SARS-CoV-2

Indronil Bhattacharjee, Jaspreet Thind

Abstract

In this study, we have implemented FrequentWords() and BetterFrequentWords() algorithms to find frequent k-mers which works on counting and hashing principle, respectively. It has been found hashing by BetterFrequentWords() performs tasks much faster as compared to Frequentwords(). To test our best algorithm, we considered multiple virus's genomes – SARS-CoV-2, SARS-CoV, MERS, Bat RaTG13 and Pangolin-CoV. In SARS-CoV-2 genome, there's poly (A) tail at the end of its genome sequence which was removed to make our results more biologically convincing. We have reported multiple length k-mer sequences in SARS-CoV-2 and some of them also found in other viral genomes. Functional annotation of genes is also reported in which candidate k-mer sequences were found from Uniprot database. No motifs were found for selected k-mer sequences using RCSB PDB motif finder.

Introduction

In the last decade due to the reduction in cost of sequencing technologies, the availability of genomics sequences becomes much more accessible due to which the number of transcriptomic and genomic bio projects has increased dramatically. The advancement in these technologies are pushing the capabilities of conventional laboratory experiments. In this study, we worked with fundamental queries of bioinformatic pipelines related to the interpretation of biological datasets which can be put together as follows: Find the k-mers of different lengths within the given sequence using different algorithms and evaluating their performance on larger datasets. Similarly, locate the motif sequences on the candidate k-mer, annotation of genes and reporting the conserved k-mers among multiple viral genomes. Since this area is well-studied, bioinformatic community has adopted many k-mer based approaches due to their potential to summarize large datasets efficiently. Hence, the algorithms we described in this study are widely accepted and imperative to biological analysis.

Methods

We used Python programming language to implement our k-mer analysis algorithms. We developed two algorithms: FrequentWords() and BetterFrequentWords() to identify most frequent k-mers in the SARS-CoV-2 genome. We also developed the FindKmerRepeats() function to identify k-mer repeats in the genome. We used a permutation test to determine the statistical significance of the frequency of most frequent k-mers. We generated null distributions by simulating sequences with the same length and ACGT frequencies as the SARS-CoV-2 genome.

TASK 1: FrequentWords()

The FrequentWords() function takes a string text and an integer k as input, and returns a list of the most frequent k-mers in text. The function first creates an empty list called freq_patterns. It then initializes a list called count with zeros whose length is equal to the number of k-mers in text (i.e., n - k + 1).

Next, the function iterates over each k-mer in text, and counts the number of times it appears in text using the PatternCount() function. The count for each k-mer is stored in the corresponding position in the count list. The function then determines the maximum count value and adds any k-

mers with this count to the freq_patterns list. Finally, the function removes duplicates from the freq_patterns list and returns it.
The PatternCount() function simply counts the number of occurrences of a given pattern in a text string, by iterating over all substrings of text of length len(pattern) and checking if each substring matches the pattern.

TASK 2: BetterFrequentWords()
The BetterFrequentWords() function is a Python function that takes a string of DNA sequence (text) and an integer (k) as input. It returns a list of the most frequent k-mers in the given DNA sequence.
This function is an improvement over the basic FrequentWords() function, as it makes use of a dictionary (freqMap) to store the frequency of k-mers in the text. This makes the function more efficient and faster for larger texts, as the frequency of each k-mer is only calculated once. The function then finds the k-mers with the highest frequency and returns them as a list.
The FrequencyMap() function is a helper function used by BetterFrequentWords(). It takes a string of DNA sequence (text) and an integer (k) as input and returns a dictionary with keys as k-mers and values as their frequency in the text. This function is called by BetterFrequentWords() to create the frequency map that it uses to find the most frequent k-mers.

TASK 3: generate_sequence()
The generate_sequence() function generates a random DNA sequence of length L. It first creates a dictionary p containing the probabilities of each nucleotide (A,C,G,T) being chosen in the sequence. Then, it uses the np.random.multinomial() function from the NumPy library to generate a list of random integers, with length L, based on the probabilities in the p dictionary. The multinomial() function returns a 2D array, where each row represents a set of values representing the outcomes of the multinomial random variable. Here, each row consists of a set of 0s and 1s representing the absence or presence of each nucleotide in the sequence. The np.argmax() function returns the indices of the maximum value in each row, which correspond to the indices of the nucleotide with the highest probability of being chosen in that position. Finally, the function generates a string s by joining the nucleotides with the highest probability at each position in the sequence.

TASK 4: Visualize the empirical runtime of FrequentWords() and Better FrequentWords() algorithms
The compare_runtimes() function takes two input arguments L and K representing the length of a sequence of nucleotides and the length of the k-mers to search for, respectively.
Inside the function, the nucleotide sequence is generated using the generate_sequence() function. Then, the function calls FrequentWords() and BetterFrequentWords() functions to find the most frequent k-mers in the sequence, and records the time it takes to execute each function using time.time().
The function returns two output values: frequent_words_time and better_frequent_words_time, representing the time it took to execute the FrequentWords() and BetterFrequentWords() functions, respectively.
The L_values and k_values variables are used to specify the range of values for L and K to use in the comparison. The function then loops through each value of L and K and calls compare_runtimes() to compare the runtime of the two functions for each combination of L and k.
Finally, the function plots two graphs using matplotlib, one showing the runtime of FrequentWords() for each value of L and K, and the other showing the runtime of

BetterFrequentWords() for each value of L and K. The graphs are labeled with the corresponding value of k.

TASK 5: Most frequent K-mers in SARS-COV-2

SARS-COV-2 is an RNA Virus. The reference genome of the SARS-CoV-2 coronavirus has 29903 nucleotides, with 33 adenine (A), which is most likely the polyA tail of the RNA sequence. We took out the polyA tail in order to discover relevant repeated k-mers. Using BetterFrequentWords() algorithm, the most frequent K-mers of SARS-CoV-2 was found for K= 3,6,9,12, and 15. The genome sequence was considered in FASTA format from NCBI database [1].

TASK 6: Biological overview of the most frequent K-mers

We downloaded the gtf annotation file [2] from NCBI for SARS-CoV-2. The genomic coordinates of selected k-mer candidates were found using the function FindKmerPositions(). Then we searched these start and end coordinates against the annotation file by which we able to locate our k-mer sequences to their gene or intragenic regions on SARS-CoV-2 genome. Function annotation was done using Uniprot database. To check the presence of k-mer sequences from SARS-CoV-2 in other viral genomes, FindKmerPositions() function was used. RCSB PDB motif finder was to locate motifs on selected k-mer sequences.

Besides the 6 tasks, we tried implementing some extra tasks like the null distribution of the frequency of most frequent k-mers, SARS_CoV-2 versus SARS_CoV [3] K-mers, finding longest K-mers beyond 15, developing a fast algorithm to find k-mer repeats, and comparison of K-mer repeats in SARS-CoV-2 with MERS [4], Bat RaTG13[5] and Pangolin-CoV[6] coronaviruses.

Results

We visualized the empirical runtime of the two algorithms by counting versus by hashing. The BetterFrequentWords() with hashing algorithm shows better and faster runtime than that of FrequentWords() with counting algorithm.

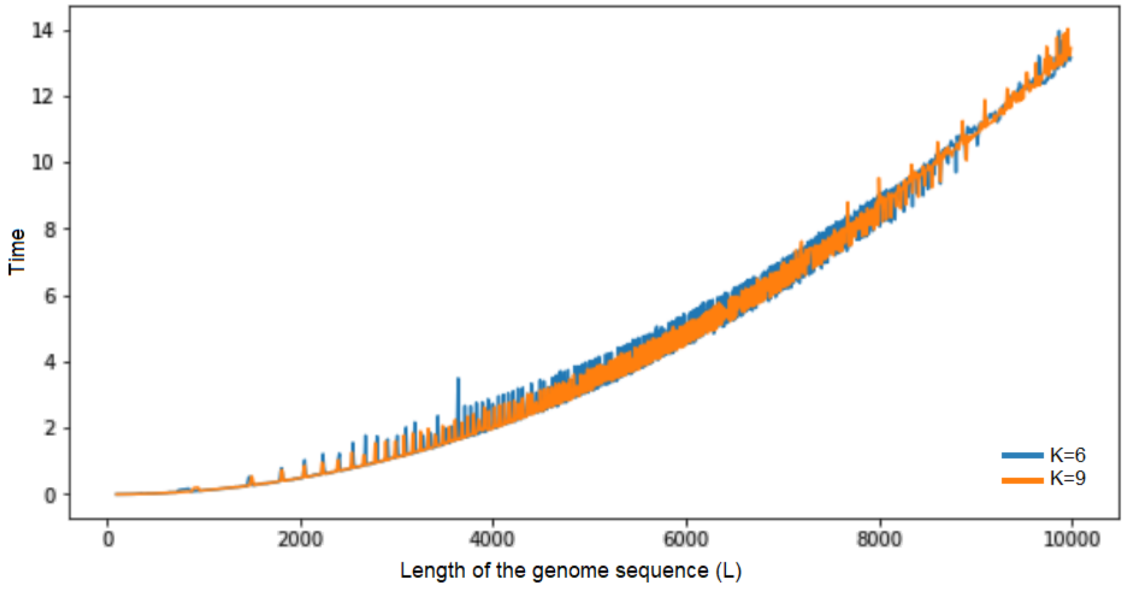

Fig 1.a: Visual implementation of runtimes of FrequentWords() algorithm

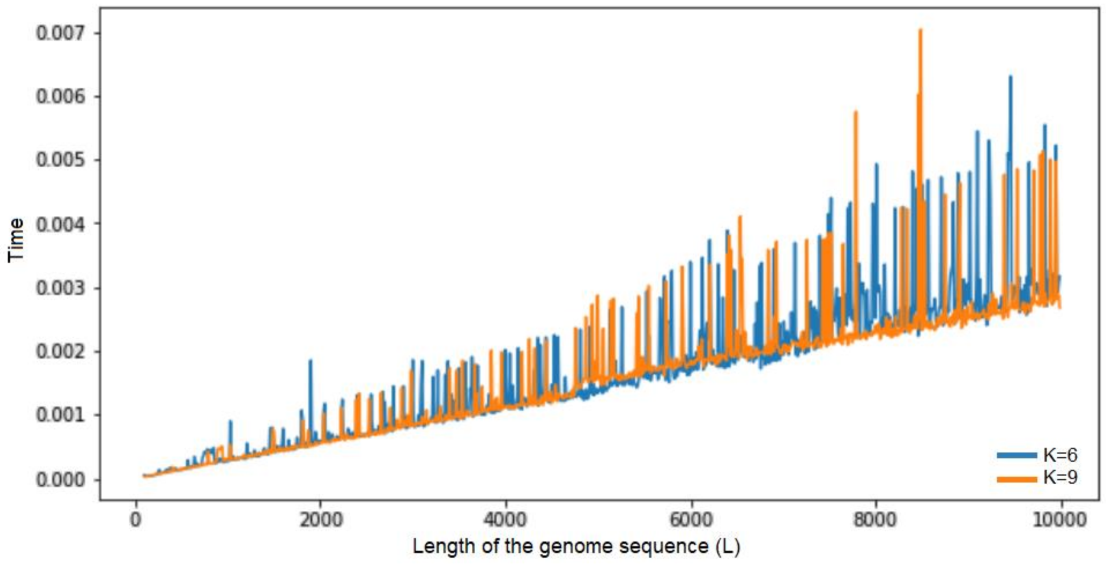

Fig 1.b: Visual implementation of runtimes of BetterFrequentWords() algorithm

Fig 1.a and Fig 1.b clearly shows the comparison and increasing of time required with increment of the length of genome sequence from 100 to 10000, with K=6 and 9.

Table 1: Biological overview of most frequent K-mers

	TCTAAACGAAC (11-mer)	TAAACGAACATGAAA (15-mer)	ATCAGACAACTACTAT (17-mer)
Number of times its appears in SARS-CoV-2	3	2	2
Gene	Intragenic region, E	First 3 nucleotides are stop codon of ORF6, 12th, 13th and 14th positions are a start codon for ORF7a	ORF1ab
Function	Currently not known	N/A	plays a pivotal role in viral mRNAs cap methylation
Presence in other species	3 times in SARS_COV_2, SARS_COV, Bat RaTG13. 4 times in Pangolin-Cov	1 time in SARS_COV, Bat RaTG13. 2 times in Pangolin- Cov	No hits
Motif	N/A	N/A	N/A

Table 1 represents the biological overview of some selected most frequent K-mers, their genetic positions, functions and presence in other species like Bat, Pangolin.

The FindKmerRepeats() algorithm shows slightly faster runtime than BetterFrequentWords() with algorithm for single K's. Fig 2.a and Fig 2.b clearly shows the comparison and increasing of time required with increment of the length of genome sequence from 100 to 20000, with K=15 and 18.

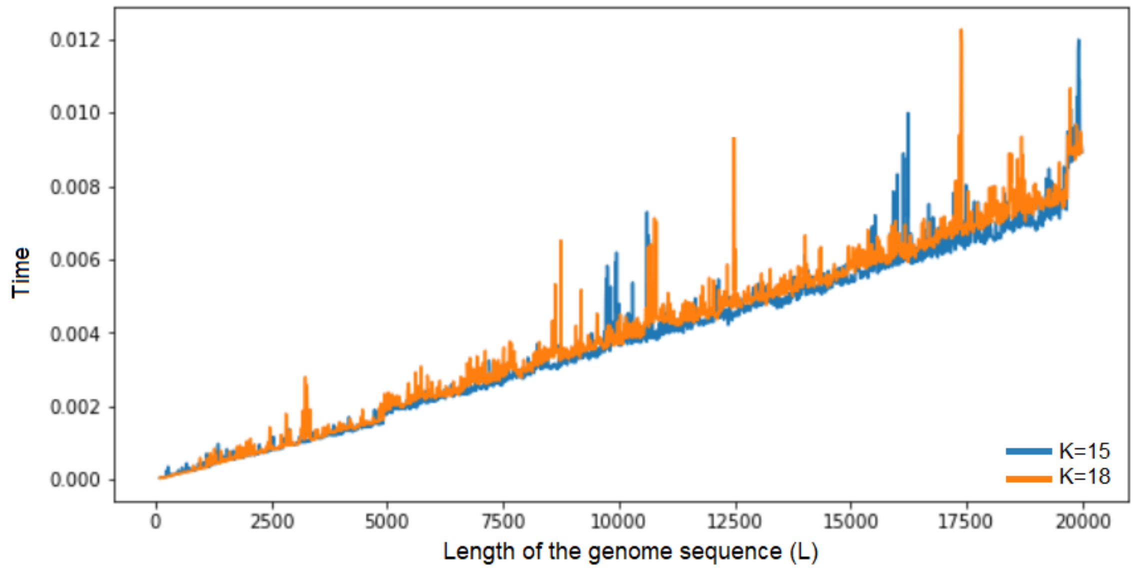

Fig 2.a: Visual implementation of runtimes of FindKmerRepeats() algorithm

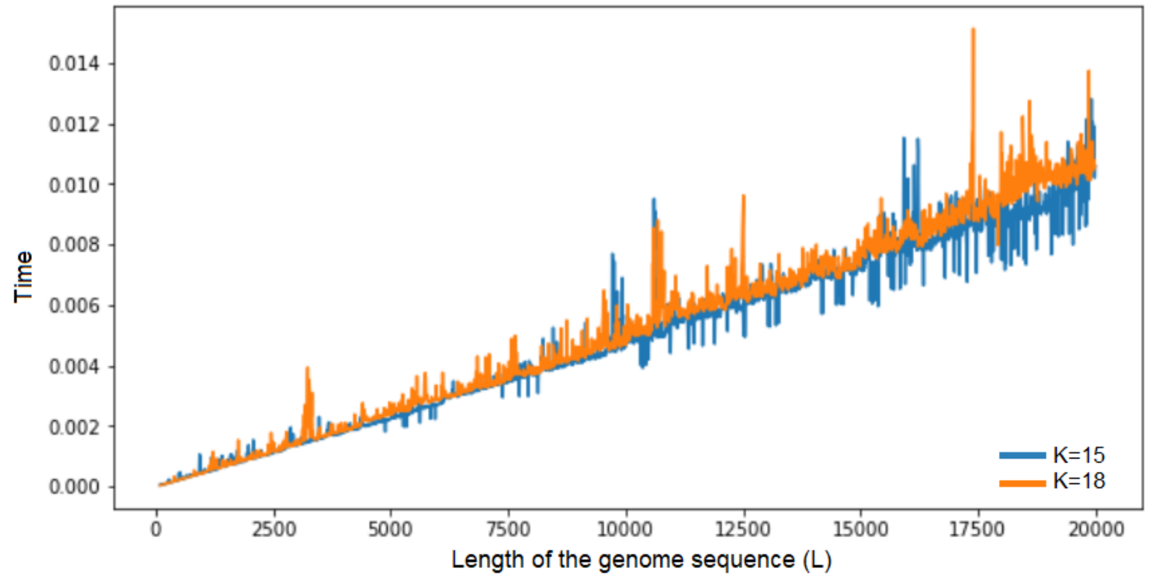

Fig 2.b: Visual implementation of runtimes of BetterFrequentWords() algorithm

Discussion

Implementation of hashing in R was a difficult task to complete. However, we find it much easier in python, and the analysis was straight forward. Finding a corresponding gene for selected k-mers also proved to be another task that took some time to complete. There was more research into new algorithms that needed to be done to make the process even more faster for k-mer analysis. During analysis, BetterFrequentwords() showed more overall good approach for k-mer identification than FrequentWords(). For sample of genome length of 10,000 nucleotides, FrequentWord() algorithm takes 0.5s to 13.9s with L increasing from 100 to 10000, where BetterFrequentWords() needs 0.003s to 0.014s. Another observation was that TCTAAACGAAC (11-mer) was appearing 3 times in SARS_COV_2, SARS_COV, Bat RaTG13 and 4 times in Pangolin. It seems biologically interesting k-mer, further analysis of this k-mer could give us more important and meaningful results. This project was a good hands-on introduction to importance of algorithms to find k-mers and how much biological importance they hold.

Distribution of Work

	Indronil Bhattacharjee	**Jaspreet Thind**
Task 1	**X**	**X**
Task 2	**X**	**X**
Task 3	**X**	**X**
Task 4	**X**	
Task 5	**X**	**X**
Task 6	**X**	**X**

References

1. https://www.ncbi.nlm.nih.gov/nuccore/NC_045512.2?report=fasta
2. https://www.ncbi.nlm.nih.gov/assembly/GCF_009858895.2
3. https://www.ncbi.nlm.nih.gov/nuccore/AY545919.1?report=fasta
4. https://www.ncbi.nlm.nih.gov/nuccore/KU740200.1?report=fasta
5. https://www.ncbi.nlm.nih.gov/nuccore/MN996532.2?report=fasta
6. https://www.ncbi.nlm.nih.gov/nuccore/MT040333.1?report=fasta