Université Grenoble Alpes

SALMONELLA OUTBREAK

Hanning Yang
Lynn Farhat
Niloufar Zarghampour

October 18, 2022
Computational Biology

Introduction

There has been a violent outbreak of Salmonella; a bacterial infection that is killing a lot of people. However, since the regular antibiotics (which contain tetracycline) are futile, our goal is to identify the root of this issue so that biologists can find a solution for it. In order to do so, we will use computational tools to compare the sequencing of the two different strains of bacteria - wild and resistant to tetracycline, identified by the biologist Emmanuelle Charpentier. The wild type of the bacterium is the one that occurs in nature and has no resistance to tetracycline, meaning it is found in the typical cases whereby the infection can be removed by taking antibiotics. The other strain is the one resistant to tetracycline, and it is the one whose differentiation we are attempting to unearth. By doing this comparison and some eliminations, we will end up with some Single nucleotide polymorphisms (SNPs). These SNPs represent the genetic variation that is potentially causing this outbreak and the resistance to tetracycline.

1 Approach

The developed tool will take as input two FASTA files, each of which contains the amino acid/nucleotide sequences of the resistant and the wild strain, whereby each of these elements is represented by a letter.The output is the detected mutations. An overview of the steps is depicted below, and in the next section we will go into details of each step.

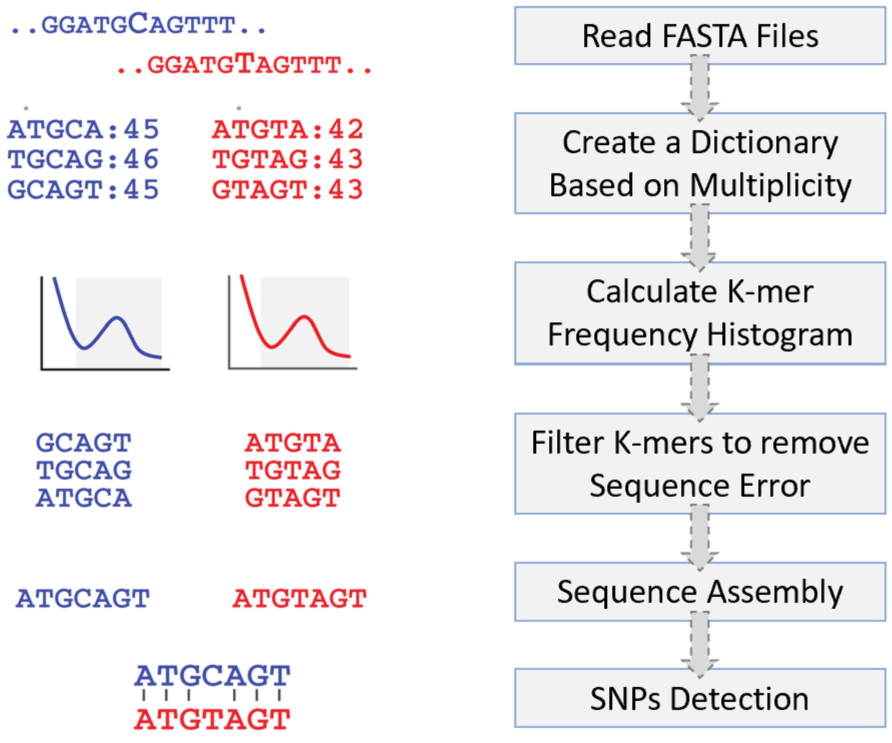


Figure 1: Workflow of the project

2 Implementation

2.1 K-mer Counting

In bio-informatics, the study of k-mers is wildly used. k-mers are substrings of length k contained within a biological sequence. Decomposing a sequence into its k-mers for analysis allows this set of fixed-size chunks to be analysed rather than the sequence, and this can be more efficient.

Now, the problem of counting the K-mers is very simple. First, we have to create a dictionary where the keys are all the possible k-mers appearing in the genome, second, for calculating the values for each key we have to count how many times that specific k-mer appears in the sequence. While this method is very simple, it is time consuming. We will investigate why. Suppose we have set k=3. Since there are only 4 nucleotides A, C, G, T, we can represent each character using only 2 bits. Let's assign 1byte / 8bits for the count. Then $8 + 6 = 14$ bits is needed per row of the dictionary. That would leave us with 4^3 rows in total. Therefore, the total memory requirement is $4^3(8 + 6) = 896$ bits $/112$ bytes. The Figure 2 shows what happens to the memory when you increase the K size.

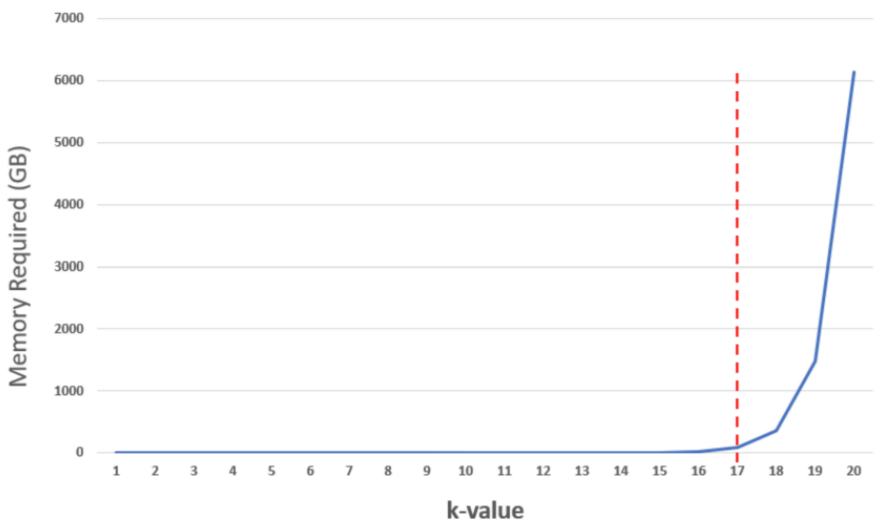


Figure 2: Memory bottleneck while counting k-mers

For k=20, almost 6000GB of RAM is needed. This memory bottleneck is what makes k-mer counting very challenging yet intriguing.

In our case, It took around 3.5 minutes to count the K-mers for each files.

2.2 Frequency Distribution of K-mers

In this section, we aim to find the best window size for the k-mer analysis and try to remove the sequence error by studying the K-mer frequency histogram and its statistical behaviour. K-mer coverage histograms are frequently used to determine the coverage cut-off for a k-mer spectrum. A k-mer coverage histogram illustrates a mixture of two distributions: one for the coverage of likely correct or "solid" k-mers and the other for spurious or "weak" k-mers. In theory, the coverage of true k-mers follows a Poisson distribution, but the biases in Illumina sequencing add variance which causes the overshoot in Figure 3.

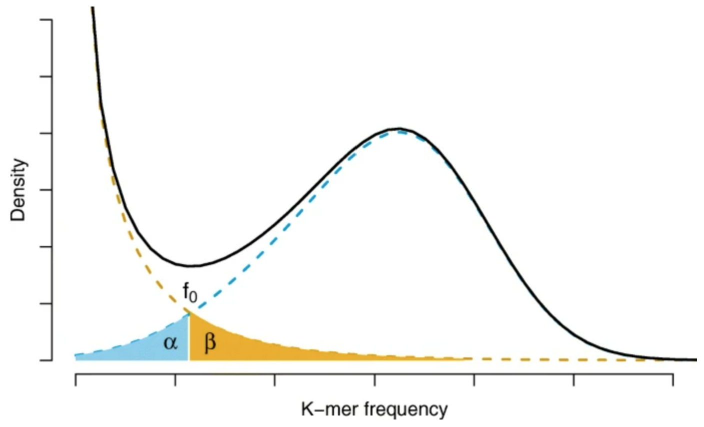


Figure 3: Frequency distribution of both error-free and erroneous k-mers: The α labeled area is the proportion of solid k-mers having frequency less than f_0, while the β labeled area is the proportion of weak k-mers

2.2.1 Choosing The Optimal Window Size

In this step we will explore the frequency behaviour of the k-mers on our dataset for different values of k in order to decide on the optimal window size .

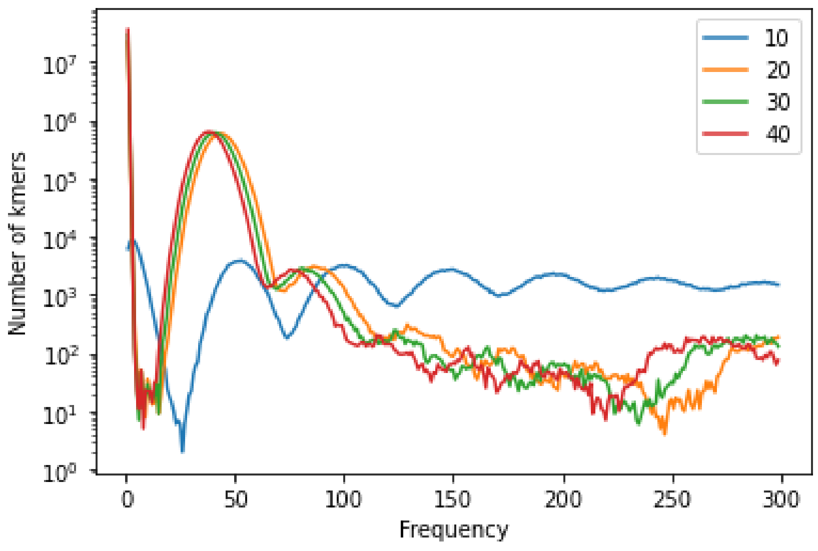


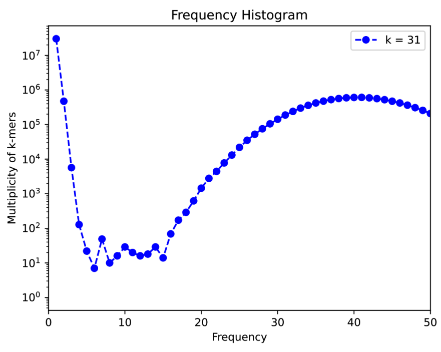
Figure 4: K-mer Frequency Histogram

As we increase k from 10 to 20, we notice that the graph takes a more definite Poisson form (minus the tails), and continues to do so after the value of k=20.
Choosing a k that is too small would result in many unrelated sequences being composed of the same k-mers, which leads to a decrease in accuracy. Moreover, this will result in other

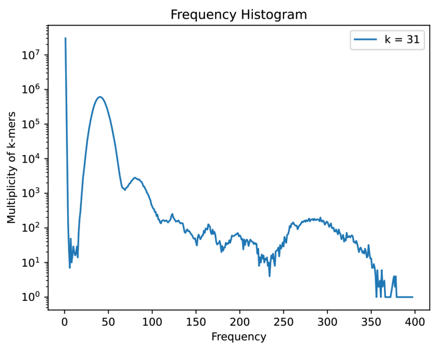
computational problems such as the memory bottleneck mentioned before. On the other hand, choosing a k that is too large would not be useful as we would not be making good use of the k-mer analysis, and the division of the sequence into few, large chunks is not very helpful.
Thus, we decided that the value of k should be some value between 25 and 35. We chose 31 since it is the largest odd value of k such that a k-mer can be translated into a 64-bit integer. Most computers nowadays can manipulate 64-bit integers very well and most programming languages will make it easy to work efficiently with 64-bit integers.

2.2.2 Removing Sequence Error

The general idea for correcting sequencing errors is that erroneous bases (i.e., nucleotides) in a DNA sequence read can be corrected using the majority of reads that have these bases correctly since errors occur infrequently and independently.
To identify erroneous k-mers in reads, we need to count the multiplicity of each k-mer. Based on Figure 3 , A k-mer is called solid if its frequency is more than f_0 , and weak otherwise. We call f_0 the **cut off frequency** that helps to filter out the weak k-mers from the solid ones. (denoted as cf in the code)
In the code, this is done by checking the multiplicity of each k-mer in the dictionary. If the multiplicity of a particular k-mer is less than cf, then it is removed from the dictionary.
Now, let's plot the frequency histogram for $k = 31$ to find this cut off frequency. Since from Figure 4 it can be seen that most of first peaks happen before $f = 50$, we will plot a histogram from frequencies 0 to 50 to find the cut off frequency.



(a) Limited range of frequencies



(b) Full range of frequencies

Figure 5: Frequency Histogram for the 31-mer

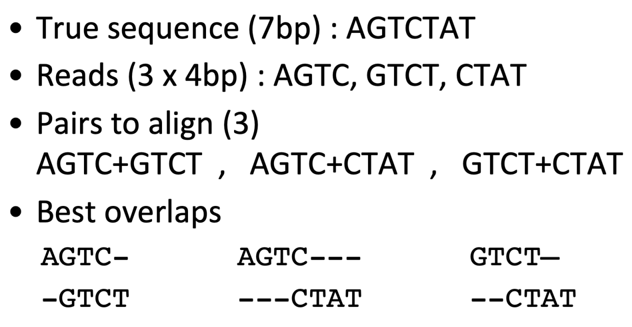
Based on Figure 5, we can clearly see that after $f = 15$, the graph starts to have the Poisson behaviour that we desire, therefore, we set cf as 15. However this is set as an input variable (it can be changed should you choose a different value of K in the beginning of the program) but the default is set to be 15.

2.3 SNPs Detection

After filtering out the erroneous k-mers, it is time to compare the two genomes to detect the mutations that results into SNPs. To do so, we need to merge the K-mers back together to assemble the genome and compare them using a distance metric called the Levenshtein distance.

2.3.1 Identify Group-specific Sequences Through Group-specific k-mer Assembly

Capturing group-specific k-mers between two groups of genomic sequences is critical for the follow-up identifications of SNPs. A k-mer that is present or rich, in one group, but absent or scarce, in another group is considered as a "group-specific" k-mer in our study.
Basically, we need to find the k-mers in the wild sequence that appear at least once in the variant sequence and remove them. We have to do the same step with the variant sequence, otherwise there will be no mutations detected.
After we captured group-specific k-mers, we assembled the identified group-specific k-mers into sequences using overlap information between k-mers. Here is an example.



- True sequence (7bp) : AGTCTAT
- Reads (3 x 4bp) : AGTC, GTCT, CTAT
- Pairs to align (3)
  AGTC+GTCT , AGTC+CTAT , GTCT+CTAT
- Best overlaps

```
AGTC-      AGTC---      GTCT–
-GTCT      ---CTAT      --CTAT
```

Figure 6: Concatention steps

We will reach sequences like Figure 7.

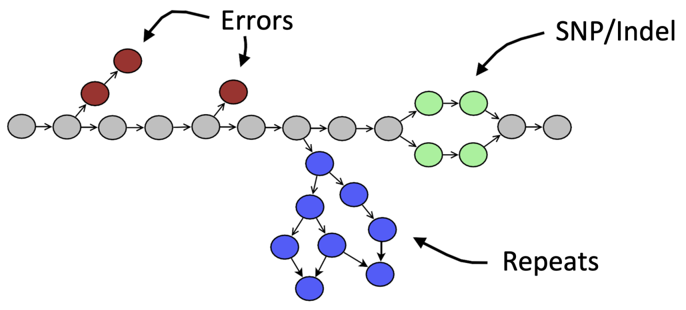


Figure 7: Result of Genome Assembly

2.3.2 Levenshtein distance

Once we removed the sequence errors and obtained the assembly for the wild and mutant types, we used Levenshtein distance as metric to compare these two sequences to detect SNPs.

Levenshtein distance is a well-established mathematical algorithm for measuring the edit distance between words and can specifically weight insertions, deletions and substitutions. For example, in order to compute the Levenshtein distance from AGTCT to GACT, the A must be deleted and the first T switched to an A, for a cost of two. This eyeball calculation only works for simple examples, so Figure 6 illustrates this perspective on the classical Levenshtein algorithm.

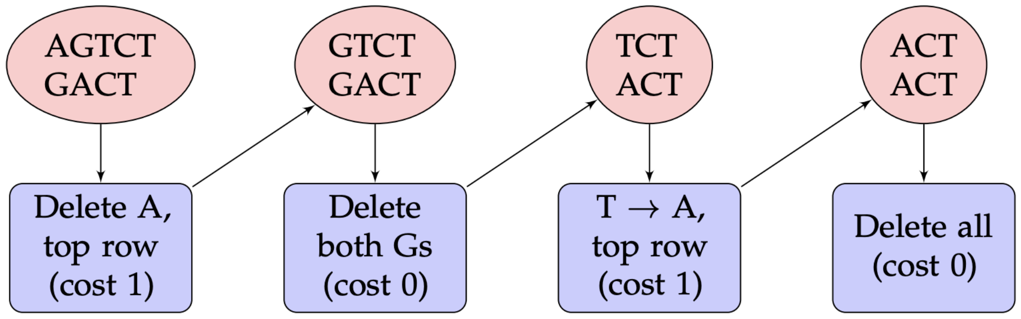
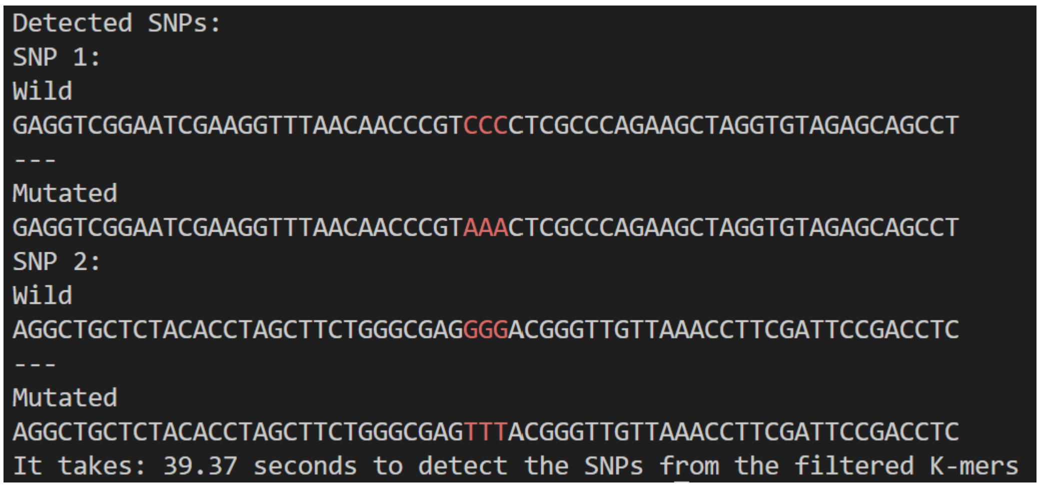


Figure 8: The steps to compute the Levenshtein distance from AGTCT to GACT

3 Result

We chose 31 and 15 as values of *k* and *cf*. The Levenshtein distance between the assembly for the wild and mutant types was obtained. Then we located the reads in both wild and mutant types, which are likely to be the SNPs. Figure 9 shows the result. We found two SNPs and the Levenshtein distance is 3.



```
Detected SNPs:
SNP 1:
Wild
GAGGTCGGAATCGAAGGTTTAACAACCCGTCCCCTCGCCCAGAAGCTAGGTGTAGAGCAGCCT
---
Mutated
GAGGTCGGAATCGAAGGTTTAACAACCCGTAAACTCGCCCAGAAGCTAGGTGTAGAGCAGCCT
SNP 2:
Wild
AGGCTGCTCTACACCTAGCTTCTGGGCGAGGGGACGGGTTGTTAAACCTTCGATTCCGACCTC
---
Mutated
AGGCTGCTCTACACCTAGCTTCTGGGCGAGTTTACGGGTTGTTAAACCTTCGATTCCGACCTC
It takes: 39.37 seconds to detect the SNPs from the filtered K-mers
```

Figure 9: Detected SNPs

Conclusions and Perspectives

In our study, we proved the coverage of k-mers follows a Poisson distribution through frequency distribution of k-mers and succeeded to detect SNPs after setting a decent k value.

Despite a promising result, there are still a lot of improvements we can work on in the future. For example, we can perform statistical analysis using a binomial or Poisson test provided by the scipy.stats Python package to test the kind of a candidate variant by determining the difference between wild-type and mutation k-mer counts in the sample. In the part of concatenation, we could reach a higher accuracy of genomes if we could show the k-mer size effect graphically, which can help us choose a better k-mer size. However, due to limitation of time and pages, it will be studied next.

References

[1] Lee, HoJoon and Shuaibi, Ahmed and Bell, John M and Pavlichin, Dmitri S and Ji, Hanlee P, *Unique k-mer sequences for validating cancer-related substitution, insertion and deletion mutations,* Oxford University Press, 2020.

[2] Rahman, Atif and Hallgrímsdóttir, Ingileif and Eisen, Michael and Pachter, Lior, *Association mapping from sequencing reads using k-mers,* eLife Sciences Publications Limited, 2018.

[3] Karl JV and Albani, Maria C and James, Geo Velikkakam and Gutjahr, Caroline and Hartwig, Benjamin and Turck, Franziska and Paszkowski, Uta and Coupland, George and Schneeberger, Korbinian, *Mutation identification by direct comparison of whole-genome sequencing data from mutant and wild-type individuals using k-mers,* Nature Publishing Group, 2013.

[4] Kelley D, et al. Quake: quality-aware detection and correction of sequencing errors. Genome Biol. 2010;11:R116

[5] Melsted, P., Pritchard, J.K. Efficient counting of k-mers in DNA sequences using a bloom filter. BMC Bioinformatics 12, 333 (2011)

[6] Yongchao Liu, Jan Schröder, Bertil Schmidt, Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data, Bioinformatics, Volume 29, Issue 3, 1 February 2013