BIOCP Project

Maréchal Lola, DNAet Chloé

October 17, 2022

1 Introduction

This project consists in implementing and computing a code which allows its user to detect genome mutations. Indeed, it gives the possibility to compare 2 sequences and to find mutations between both and details about it. In order to solve the problem we use python libraries and modules to extract data and balance information.

2 Strategy

First we have decided to understand *how to extract data* from input files. The fasta file contains 1993166 reads (250bp) of a genome (5Mbp). The complexity issue advises us to work with k-mer. A k-mer is a sequence of $\boldsymbol{k}$ amino acids. The first step of our plan is to build a dictionary of k-mer of each fasta file in order to simplify mutation detection. Nevertheless, the optimal k-length is unknown. So our first step will be to find the best k : not too big and not too small. Indeed, if k is too big there are many sequences reading errors (because of the 1% of errors of reading). However, if k is too small the k-mer has more chances to be found too much times at locations where it doesn't mean this DNA sequence is specific and the one we look for.

In addition : as there is a huge quantity of data there are several reading errors. Considering that proportion of reading errors for a nucleotid is 1%, we will clean our data. In order to clean the dictionary previously build : we will use the Poisson distribution. It will allow us to find a LEVEL. This level corresponds to the number of k-mer occurrences in the file under which we won't keep the k-mer because it has many chances to contains reading errors. Both healthy and mutated dictionaries will be cleaned this way : we only keep $\boldsymbol{solid\ k-mer}$.

Finally, both dictionaries with solid k-mers will be rebuilt and compared in order to find matching k-mer, SNPs and their context. Indeed it is necessary to rebuild following sequences in the dictionary that is to we must gather the consecutive k-mers and find matches between both dictionaries.

3 Execution of the strategy

3.1 Find optimal k for k-mer length

3.1.1 Steps

1. Extract a fixed (500) number of k-mer in the original sequence with iterable data. These k-mers are from different reads of the file. Build a dictionary with k-mers as keys and their occurrence numbers as values.

2. Read all the strain and count the number of occurrences for each k-value. Indeed, for each k-mer if it is recognised in the dictionary : we add 1 to the dictionary value which match with the key $\boldsymbol{k-mer}$. Else, if this k-mer is not recognized : we do nothing.

3. Finally, build histograms for each k value tested which represent the number of occurrences for each sequence chosen and find the best k to use.

3.1.2 Observations

Keeping in mind that our file contains on average 100 ($1993166 * 250/5000000$) iterations of the same genome, we would like to find a mean of 100 occurrences for all k-mer in the implemented dictionary. On the one hand the point is that if the k-mer is too large the probability to read a lecture error increases. In addition, it's possible that this expected k-mer finish at the beginning of a new read. On the other hand, if the k-mer is too small it could be read too many times even where the sequence is not really the one we wanted to read.

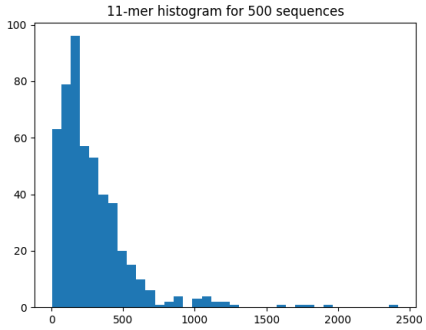

Figure 1: Histogram of 11-mer occurrences in the input strain

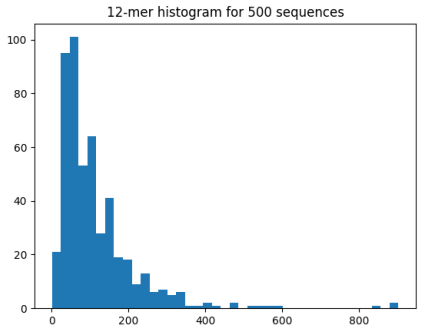

Figure 2: Histogram of 12-mer occurrences in the input strain

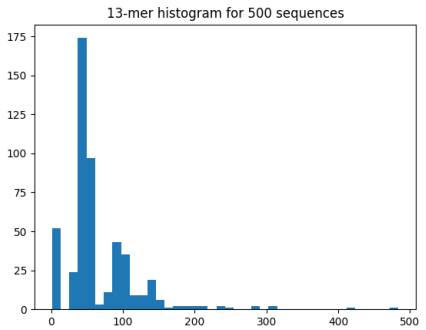

Figure 3: Histogram of 13-mer occurrences in the input strain

Once we have computed our code with different k values for k-mer length, we made interpretations with output values.

The results are (with 500 sequences)

- For $\boldsymbol{k = 11}$ mean = 284.704, median = 214.0
- For $\boldsymbol{k = 12}$ mean = 114.752, median = 89.0
- For $\boldsymbol{k = 13}$ mean = 65.362, median = 49.5

Balancing our expectation and results, we are able tot take a decision for optimal k. Both means for case $k = 11$ and $k = 13$ are too low and to high. The case $k = 11$ mean is too high and it means the k length is too small and the probability to find a k-mer in a wrong read place is high. Moreover the case $k = 13$ mean is too low and it might be too hard to find a write k-mer without moving on the next read. Thus, we choose the case $k = 12$ with $median = 89.0$ which is lower than 100 due to the 1% error.

3.1.3 Dictionary construction

Once the optimal k-mer length has been found: the dictionary containing all k-mer from the fasta file are build. In fact, both healthy and mutated files are used to build 2 dictionaries. They has all different k-mer sequences as keys and their number of occurrences in the whole file as values. Our strategy is to browse each entire file in reading all reads from it and in loading each new k-mer as new key and add 1 to the matching value for each already seen k-mer.

3.2 Poisson distribution

3.2.1 Goal

In order to compare both reads, we first need to remove all the k-mer with reading errors. To realize this tasks, we modeled the probability to overlap a sequence in all the read and the number of overlapping and the probability to find reading errors. Using this modelling it is possible to use Poisson distribution in order to find a value above which the k-mer is removed because it contains reading errors.

3.2.2 Model

Let X the random variable representing the number of reads overlapping a given k-mer. It follows a binomial distribution with parameters N and p where N is the number of reads of the fasta file and p the the probability of the read to overlap a given k-mer. As N is really big, we can make the approximation that X follows a Poisson distribution with parameter $\lambda = N \times p$.
Here, $p = \frac{r-k}{G-r}$ with r the length of a read and G the size of the genome.
Then, let Y the random variable representing the number of reads that overlap a k-mer that contains an error. As for X, we can approximate that Y follows a Poisson distribution with parameter $\lambda * \frac{\epsilon}{4}$ where ϵ is the frequency of reading error (0.01) and 4 because there are 4 nucleotides.

3.2.3 Results

Here is the histogram we obtain:

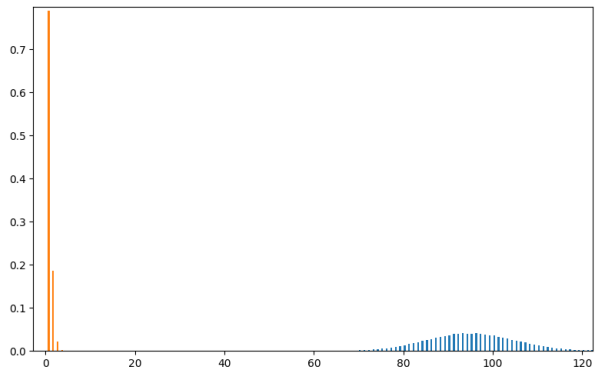

Figure 4: Histogram of X (blue) and Y (orange)

We need to chose a level which is between both distributions, to determine below how many occurrences a k-mer is considered to be due to a reading error. According to the histogram, we have decided to chose 35.

3.3 Comparison of the genomes

3.3.1 Initialisation of the dictionaries

First, we call the function *init_dict()* which initialized two dictionaries (one for each genome) containing all the k-mer of the sequences (with the k we have chosen previously) as keys, and the number of occurrences of the k-mer in the sequence as values.
After that, the same function *init_dict()* calls on the two dictionaries the method *clear_dict()* which removes the k-mers containing reading error, that is to say those which appears less than 35 times in the sequences.
Finally *init_dict()* returns the 2 dictionaries with "$\boldsymbol{solid\ k-mers}$".

3.3.2 Comparison

The comparison step consists in finding the differences between the two dictionaries (healthy and mutated).
To do that, we implemented the function *compare_dict()*. It takes the two dictionaries as arguments, and browses all the keys of one of them, and removes it from the second one if the sequence is also in it. Else, if it's not present, it puts the k-mer in a list. Indeed, if this k-mer is not present in the other dictionary: as it is a solid k-mer it means that it is a mutated sequence. Finally, the function returns the list of the k-mer present on the first dictionary and not on the second, and the second dictionary without the k-mer which were present on the first one.
Then, we convert the dictionary into a list of those k-mers, we do not need the occurrences anymore.

3.4 Alignment and comparison

Now that we get 2 lists with mutated k-mer and healthy corresponding k-mer, the goal is to find matches between both. In order to do that, we will use *blastx*.
This tools allows its users to realize the alignment and to find the mutations between a list of sequences containing mutation errors and a reference fasta file.
Indeed, it computes alignment and also comparison between different sequences. It exists different options from blast. BLASTx is one of the three BLAST options. In BLASTx: a nucleotide sequence

is used as a query. This query is first translated in all six reading frames and after that each of the amino acid sequences is compared to the protein sequences built: here it is named *salmonella-enterica.reads.fna*.

Before using this tool, we have created a fasta file using the function *write_fasta* with sequences from the list of mutated sequences. Thus we should use the web tools blastx with both files : our reference *salmonella-enterica.reads.fna* and the built file all along our plan. This tasks took too much time to compute so we didn't used Blastx even if we compute all the other part until the creation of our *mutated.fasta* which is our final file with solid and mutates k-mers.

4 Time and complexity

The initial problem is to find an existing mutation between two files containing genome readings. We have two .fasta files, one "healthy" and the other "mutated". The naive solution to the problem is to go through all the reads one by one and compare them to all those in the other file and find similar sequences and then determine what mutation is present.

Considering the file sizes, this solution would be very expensive (quadratic complexity) and would take too much time.
Our k-mer strategy has been chosen in order to decrease complexity. The sequence frequency of k-mer in the genome can be used as a sub-sequence. Comparison of these frequencies is mathematically easier than sequence alignment. It can be used as a first step of analysis before an alignment.

In addition to that we made choices in order to decrease our algorithm complexity as our decision to use dictionary. Indeed, it is less expensive to browse and delete items within it than in a simple list.

Nevertheless, the course of a file is very long because even if it is linear, both files take up a lot of space. We implemented different functions in which we used conditions and loop thus the time taken is important. Finally the computation of all our strategy takes several minutes, but it ended and is less expensive than naive strategy.