Computational Biology - Salmonella Outbreak

Alessandro Bonadeo, Giuseppe Intilla, Sofia Moroni

October 18, 2022

1 Introduction

In this work we developed a software to find single-nucleotide polymorphisms (SNPs) given two DNA sequences, one of which presents mutations. SNPs are a single nucleotide substitution at a specific position of the genome. In this particular task, which has been commissioned by TATFAR, the goal is to find the mutation of Salmonella bacteria's genome that is resisting to common antibiotics, causing many deaths. This phenomena is known as AMR (antimicrobial resistance), and in this particular case we're facing an antibiotic resistance that is a major subset of AMR. Finding part of genome responsible for the resistance should hopefully be an important starting point to develop antibiotics that will be able to face this new dangerous Salmonella mutation.

2 Problem analysis

We start by analyzing the problem in several aspects, paying attention to the underlying statistical nature of the problem and its link with the given data. This will give a more clear view on the solution we choose to implement.

2.1 Provided data

Two DNA sequences of the bacteria are provided: the first one is the wild type while the second one is the variant. Both are in the Fasta format, which is a text-based format for representing nucleotide sequences using single-letter codes. The entire genome of the Salmonella bacteria is around 5 millions of nucleotides but we don't have it as a unique sequence. We're given different reads of 250 nucleotides, extracted in such a way that every part of the genome is *covered*. This way of sequencing DNA is called *shotgun sequencing* and in our case is performed by Illumina machine. As stated above, we can assume that every nucleotides of the original sequence are represented in at least one of the reads. Usually, as in our case, every part of the genome appears more than one time in the extracted reads. The average number of reads representing a given

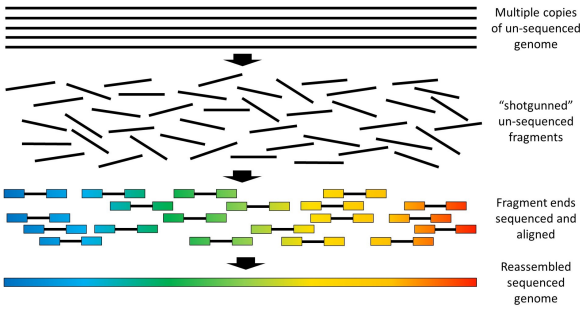

Figure 1: Shotgun sequencing

nucleotide is called *coverage* (or *frequency depth*) and is computed as:

$$C = \frac{N \times L}{G}$$

where N is the total number of reads, L the length of each read and G is the length of the original genome. In our case we have: G $\approx 5 \cdot 10^6$, N $\approx 2 \cdot 10^6$, L = 250 and then C ≈ 100 meaning that, for every nucleotide in the genome, we can expect around 100 overlapping reads containing it. Another important point to take into account is that the shotgun sequencing is not free of error: every nucleotide has a probability of 1 out of 100 of being an error. In other words there is a uniformly distributed transcription error of 1% and we should expect around 2.5 error per read.

2.2 Statistical analysis of the problem

In order to perform the analysis on the two sequences we have extracted *k-mers* which are all the possible substrings of length k of a sequence. If we consider the probability of a given k-mer of being contained in a read, this is given by:

$$p = \frac{L - k + 1}{G - L + 1}$$

which would be the number of possible position a k-mer can take place in a read over the total number of possible position a read can occupy in the genome. Now defining the random variable X as the number of time a given k-mer is found in the different reads (i.e the overlap number for a k-mer), it's immediate to say that

$$X \sim \text{Binomial(N, } p)$$

Furthermore, remembering the approximation for a binomial distribution when $p << \text{N}$, we can assume

$$X \sim \text{Poisson}(\text{N} \cdot p)$$

2.3 Choice of k and errors detection

A way to visualize X is to consider the k-mer spectrum which shows the multiplicity of each k-mer versus the number of k-mers with that multiplicity.
The goal is to find the right k such that the empirical distribution (the one extracted from the data) matches in an approximate way the theoretical one (derived in the paragraph above). In general we can say that the different sizes of k has both positive and negative effects. For example: larger values of k are more specific and accurate (since a bigger portion of genome is taken) but at the same time they will present more sequencing errors. After several tests we find that choosing a value for k between 50 and 70 leads to almost the same result.

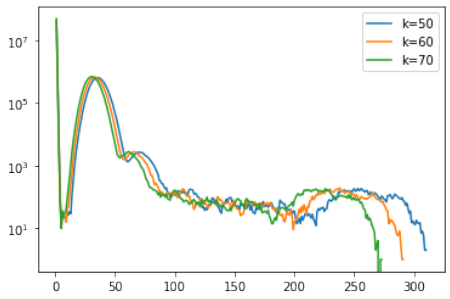

Figure 2: k-mer spectra for different values of k with log-scale on y axes

So we decide that $k = 50$ should be a valid choice for our purpose.
After that, we can observe in fig. 3 an unusual peak around the small frequencies. This is due to the fact that sequencing errors distort the correct shape of the distribution. Indeed it's highly unlikely to have the same error repeated for more than one k-mer and so each of these occurrences will have a relatively small count. Then we introduce another parameter f_{min} which will represent the smallest value of frequency that is going to be considered error free: each k-mer presenting less then f_{min} occurrences won't be used in the main algorithm. We decide to cut the distribution after the first minimum and so we empirically choose f_{min}=11.

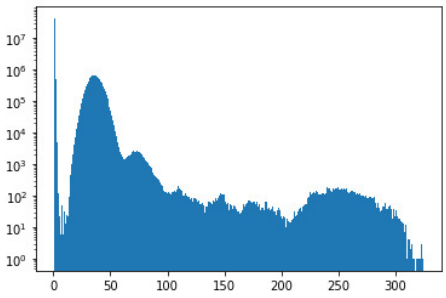

Figure 3: Histogram of 50-mers frequency with log-scale on y axes

3 Algorithm

The software requires as input two DNA sequences in FASTA format and processes them through the library BioPython. At first we decide to divide the reads into k-mers in order to exploit their statistical properties and to overcome the problem of the sequencing errors. Our function counts the frequencies of all the k-mers contained in the two sequences given a fixed k. At this point we have a dictionary for both sequences containing the k-mers and the correspondent frequency. Justified by the theoretical deductions done before we try to delete the effects of sequencing errors by cleaning our data structures:

- first we remove all the k-mers that have less than f_{min} occurrences, since they are probably affected by error;
- then we delete from the two dictionaries of k-mers all the copies, since the mutation is certainly not present there.

This process helps us to delete all the useless data since we end up with about 150 k-mers for every sequence. Now we have a tractable quantity of data and we could directly compare the actual k-mers to find the SNPs. But, since the final request is a 100 long sequence containing the mutation we try to do genome reconstruction, taking clue from [3]. We concatenate the k-mers in order to obtain bigger strings and to easily find the mutation by direct comparison. So we define a function that, for both sequences, passes through the remaining k-mers looking for overlaps. For example, given $k = 5$, let s_1, s_2 and s_3 be three k-mers:

$$s_1 = AGACG \qquad s_2 = GACGT \qquad s_3 = TAGAC$$

we end up with these two possible concatenated sequences

$$s_{1,2} = AGACGT \qquad s_{1,3} = TAGACG$$

Then we iterate this procedure and we check a possible overlap between the new sequence $k + 1$ long and other k-mers until no more matches can be found. Here we present the pseudocode for this function.

for s in sequences **do**
  for k1,k2 in kmers **do**
    if $find\_match(k1, k2) == true$ **then**
      $k1 = merge(k1, k2)$
      $delete(k2, kmers)$
    end if
  end for
end for

As a result we obtain a list for the wild strain and one for the variant type containing less than 10 sequences. In order to find the mutation we try to measure the similarity between our strings. In particular we use the Levenshtein distance [1], a metric that indicates the number of edits (insertion, deletion or substitution) between two words. We compute the distance matrix for the two lists and we expect to find the mutation between the sequences with the shortest distance.

4 Results

Here we present the results of our experiment using $k = 50$ and $f_{min} = 11$ as parameters. As we can see in the figure we end up with two possible mutations consisting of a sequence of 3 changed nucleotides. In particular in the first result a triplet of thymine is converted to guanine and in the second one adenine is exchanged with cytosine.
Finally, we use the BLAST [4] software to link the protein to the output sequences: we find that they correspond to the group called TetR (Tet Repressor) which are proteins responsible for giving antibiotic resistance to large numbers of bacteria.

```
First SNP

WILD:        TTACATGCCAATACAATGTAGGCTGCTCTACACCTAGCTTCTGGGCGAGTTTACGGGTTGTTAAACCTTCGATTCCGACCTCATTAAGCAGCTCTAATGCG
VARIANT:     TTACATGCCAATACAATGTAGGCTGCTCTACACCTAGCTTCTGGGCGAGGGGACGGGTTGTTAAACCTTCGATTCCGACCTCATTAAGCAGCTCTAATGCG


Second SNP

WILDS:     CGCATTAGAGCTGCTTAATGAGGTCGGAATCGAAGGTTTAACAACCCGTAAACTCGCCCAGAAGCTAGGTGTAGAGCAGCCTACATTGTATTGGCATGTAA
VARIANT:   CGCATTAGAGCTGCTTAATGAGGTCGGAATCGAAGGTTTAACAACCCGTCCCCTCGCCCAGAAGCTAGGTGTAGAGCAGCCTACATTGTATTGGCATGTAA
```

5 Conclusion

In conclusion our software answers to the TATFAR call finding the presence of mutated sequences in the genome. Regarding the complexity of the algorithm we can state that it is at most linear with respect to the dimension of the genome. Indeed in the first part of the experiment, exploiting the constant cost of accessing elements in dictionaries, we use functions with a complexity of $\mathcal{O}(G)$. After that we reduce the dimension of our data to $n \approx 150$ by eliminating useless k-mers and we implement functions (concatenation step and similarity

computation) that have a complexity of $\mathcal{O}(n^2)$. Since $n << G$ we can assume a global linear complexity.

References

[1] Ethan Nam, *Understanding the Levenshtein Distance Equation for Beginners*, 2019

[2] Pierre Peterlongo, Nicolas Schnel, Nadia Pisanti, Marie-France Sagot, Vincent Lacroix, *Identifying SNPs without a reference genome by comparing raw reads*, 2010

[3] Atif Rahman, Ingileif Hallgrímsdóttir, Michael Eisen, Lior Pachter,*Association mapping from sequencing reads using k-mers*, 2018

[4] Basic Local Alignment Search Tool, BLAST

[5] Code available here.