MAPPER - USER REPORT

January 11, 2021

Programs : `index.py` / `map.py`

Quick explanation of the programs

These programs are used to map a set of read on a genome indexed by FM-index method with a **seed-and-extend** approach : a read is cut in k-mers of size k (chosen by the user) and the occurences of the k-mers are found in the genome thanks to its FM-Index (complexity O(k) for searching a k-mer in the FM-Index).

For each position occurence of the k-mer found, the read is mapped on the genome at this position and the mismatches are counted. The number of maximum mismatches admitted is also a parameter chosen by the user (h_value). The mapping position retained for a read is the one that is on the lowest index on the genome and that have the less mismatch possible.

We first try to map a read on the forward strand and if an optimal match is not found, we try it on the reverse strand.

So first, the reference genome must be indexed using `index.py` and then the reads will be mapped using `map.py`.

This program counts the Single Nucleotid Polymorphisms detected during mapping and store them in a VCF file.

These programs were developed in python version 3.7 and are only executable with python version higher than 3.0

Authors : Elodie GERMANI, Kevin CHATEAU

1 Indexation

1.1 File to use : index.py

1.2 Options :

- `--ref` or `-r` + filename
  - Indicates the file containing the genome/sequence to index.
  - The file must be in fasta format : starts with a line explaining the file (will not be used) and the rest of the file contains the sequence.
  - **WARNING** : Sometimes, the fasta file containing the sequence ends with tabs or line feeds, delete them before using the file.
- `--out` or `-o` + filename
  - Indicates the output file in which we will store the index.
  - The name must be “name.dp” since the index Object is stored in a dumped file.

- `--help` or `-h`
  - To visualise this README notice.
- `--verbose` or `-v`
  - To print the options of the program and the time taken by the program.

1.3 Usage :

```
python3 index.py --ref reference.fasta --out index.dp
```

1.4 What it does :

- Takes a genome stored in a fasta file and indexed it with FM-Index method.
- Stores the BWT (Burrows Wheeler Transform) of the genome sequence, the F column, the SA (Suffix Array), the n Dictionnary and the r list.
- Will be used to map reads on the genome.
- Can also be used independantly from the other file : contains a class FMI that index a genome and methods to search for occurences of a pattern in that genome.
  - For example :
    ```
    genome_file = open(reference_file, 'r')
    genome = genome_file.readlines()[-1]
    genome_FMI = FMI(genome)
    genome_FMI.get_occurences("ATGCATGCATGC")
    ```
    will return the list of occurences positions of the pattern "ATGCATGCATGC" in the genome.

2 Mapping

2.1 File to use : map.py

2.2 Options :

- `--ref` or `-r` + filename
  - Indicates the file containing the genome/sequence to index.
  - The file must be in fasta format : starts with a line explaining the file (will not be used) and the rest of the file contains the sequence.
  - **WARNING** : Sometimes, the fasta file containing the sequence ends with tabs or line feeds, delete them before using the file.
- `--index` or `-i` + filename
  - Indicates the file in which we stored the index.
  - The name must be "name.dp" since the index Object is stored in a dumped file.
- `--reads` or `-r` + filename
  - Indicates the file containing the reads to map on the genome.
  - The file must be in fasta format : for each read, we have a line starting with ">" that names the read and after that a line containing the read.
- `--out` or `-o` + filename
  - Indicates the file in which we will store the results of our mapping.
  - The name must be "name.vcf".
- `--k_value` or `-k` + number
  - Indicates the value used to cut the read into k-mers of length k.
  - The value must be between 1 and the length of the read.

- `--max_hamming` or `-h` + number
  - Indicates the value used to map the read in the genome with a number of substitutions inferior to h.
  - The value must be between 1 and the length of the read.
- `--min_abundance` or `-m` + number
  - Indicates the value used to count the single nucleotid polymorphisms.
  - Only SNPs listed more than m times will be recorded.
- `--help` or `-h`
  - To visualise this README notice.
- `--verbose` or `-v`
  - To print the options of the program and the time taken by the program.

2.3 Usage :

```
python3 map.py --ref reference.fasta --index index.dp --reads reads.fasta -k 20
--max_hamming 5 --min_abundance 1 --out snps.vcf
```

2.4 What it does :

- Takes a genome stored in the **reference file** and indexed using index.py program and map a set of reads on it using **seed and extend** method.
- For each read, cuts it into **k-mers** of size k, search the occurences of the k-mer in the genome using its index and try to map the read at this position by counting the **substitutions** :
  - If the number of substitutions is higher than h value, the read is not considered as mapped at this position.
  - If the number of substitutions is lower than h value and lower than the number of substitutions found during the mapping of the read at a smaller position index, the read is considered as mapped at this position.
- For each read mapped on the genome, the program stores the substitutions found. At the end, when all reads were tried and mapped (or not), the substitutions are listed and abundances of each substitution are stored.
  - The program returns a **VCF file** containing the list of Single Nucleotid Polymorphisms (substitutions) that are detected more than m times.

2.5 Example of VCF file

```
#REF: reference.fa  #READS: my_reads.fa  #K: 14  #MAX_SUBST: 5  #MIN_ABUNDANCE:
10  129836 A T 26  145831 C G 25
```

3 Results

To test our programs we used a dataset containing : - the 150 000 first nucleotids of the genome of the bacteria Escherichia coli in a file `ecoli_sample.fasta` - a read file `ecoli_mutated_reads_1000.fasta` in fasta format in which each read is on a line - these reads come from a mutated version of the genome of E. coli distant from 1000 substitutions, 2 insertions and 2 deletions. - they cover 20 times the genome - there is a sequencing error rate of 1%

We wanted to see if the k_value, max_hamming and min_abundance values changed the **time of execution** of our program, the **memory peak** or the **quality of the results**. We had a

verification file and a verification script to compute the precision of our analysis and the recall.

We tested it with several values for k_value (k), max_hamming (h) and min_abundance (a) and obtained the results showed in Table 1.

Before that, we tried it with the extreme value of 1 for min_abundance. With this value, we obtained a good recall value but a **precision rate lower than 5%**. This is due to the fact that we select too many SNPs that are actually not SNPs : these are **sequencing errors**. Indeed, when a substitution is found only once, it is more likely that it corresponds to a sequencing error. Thus, we decided to find our optimum values in k_values 10, 15, 20, 25 or 30 / h_values 3, 5, 10 or 15 and min_abundance 3, 5 or 10.

We can see that when the k_value increases, the **time of execution** increases also. It is due to the alignment method chosen here : the seed and extend strategy has a complexity **O(k)** and this is well illustrated here. For the **memory peak**, there is not a real pattern here, values seems kind of random.

For the **quality of the results**, we can see that when min_abundance = 3, the precision rate and the recall are a little bit lower than whith higher values. This difference is not really significant. We can also see that when we choose a high value for h_value and a small value for min_abundance, the recall is lower than with other values (83% when h_value = 10 and min_abundance = 3).

In conclusion, we chose values for which the ratio time of execution / precision is respected. To our opinion, the optimal values are **k_value = 20, h_value = 5 and min_abundance = 5**.

k_value	h_value	min_ abundance	Time of execution (s)	Memory Peak (kB)	Precision (%)	Recall (%)
10	3	3	84.5	18569	99	97.8
10	3	5	84.5	63900	100	95.3
10	3	10	96.7	363316	100	83.2
10	5	3	85.7	221916	98.1	99.5
10	5	5	90.3	127496	100	99.4
10	5	10	92.9	215436	100	98.2
10	10	3	95.8	332572	97.7	99.5
10	10	5	95.3	119964	99.8	99.4
10	10	10	97.7	236792	100	98.7
10	15	3	99.09	141548	93.3	99.5
10	15	5	97.01	212448	97.8	99.5
10	15	10	98.5	87444	100	98.7
15	3	3	108.5	4572	99	97.8
15	3	5	111.2	72044	100	95.3
15	3	10	108.07	101164	100	83.2
15	5	3	103.9	92092	98.1	99.5
15	5	5	106.3	528324	100	99.4
15	5	10	102.4	24576	100	98.2
15	10	3	107.3	44488	97.7	99.5
15	10	5	102.4	44040	99.8	99.4
15	10	10	102.4	60340	100	98.7
15	15	3	102.7	116320	93.7	99.5
15	15	5	101.1	78432	97.8	99.5

k_value	h_value	min_ abundance	Time of execution (s)	Memory Peak (kB)	Precision (%)	Recall (%)
15	15	10	100.1	29244	100	98.7
20	3	3	116.1	224480	99	97.8
20	3	5	111.0	57428	100	95.3
20	3	10	100.6	140668	100	83.2
20	5	3	97.7	49744	98.1	99.5
20	5	5	97.9	33352	100	99.4
20	5	10	103.6	86940	100	98.2
20	10	3	100.6	36848	97.7	99.5
20	10	5	99.9	44508	99.8	99.4
20	10	10	98.8	71560	100	98.7
20	15	3	98.8	37364	93.7	99.5
20	15	5	97.4	44248	97.8	99.5
20	15	10	99.1	68404	100	98.7
25	3	3	107.9	88100	99	97.8
25	3	5	106.3	36280	100	95.3
25	3	10	104.6	71484	100	83.2
25	5	3	112.2	360920	98.1	99.5
25	5	5	115.2	45900	100	99.4
25	5	10	102.9	151864	100	98.1
25	10	3	115.5	174100	97.7	99.5
25	10	5	101.7	37156	99.8	99.4
25	10	10	102.8	95464	100	98.5
25	15	3	100.8	51516	93.7	99.5
25	15	5	102.5	40852	97.8	99.5
25	15	10	108.5	580568	100	98.5
30	3	3	114.8	31092	99	97.7
30	3	5	117.1	101420	100	95.2
30	3	10	116.5	79264	100	83.1
30	5	3	117.9	412164	98.3	99.5
30	5	5	127.9	149576	100	99.3
30	5	10	124.6	168372	100	97.7
30	10	3	129.8	391640	97.9	99.5
30	10	5	128.2	206460	99.8	99.3
30	10	10	131.5	40696	100	98.1
30	15	3	131.9	741796	93.9	99.5
30	15	5	133.4	889304	97.8	99.4
30	15	10	131.8	257536	100	98.1

Table 1 - Results obtained during the execution of map.py with different parameters.

The results obtained with the method used here (seed and extend) are highly dependent on parameters chosen :

- if the k_value chosen is too low, we will have more chances to map the read on the genome (because a small k-mer have often more occurences on the genome than a large k-mer, so we

will try more position to map the read) but if we also choose a high value for h_value, we take the risk to map the read at a wrong position.

- however if the k_value chosen is too big, we will try less position to map the read so we might miss the right position : for example, if we have a read of size 100 with 1% of sequencing errors and for example 4% of substitutions : we need k-mers of a size lower than 100/(1+4) = 20 to be sure to seed the read.
- in addition, the h_value is important because if we choose a small h_value, we could miss the right position because of a high proportion of substitutions or sequencing errors. We recommend to choose a h_value of approximately the number of substitutions+sequencing errors expected in the read. In our case, it is approximately 5.
- for the value of min_abundance, as we said before, it is important to discriminate if a SNP is a sequencing error or a mutation. A value of 1 is deprecated. A high value will also not be recommanded : we could miss mutations.

4 Covid Data Analysis

With the parameters settings we made when studying E. coli datas, we analysed the **SARS-CoV2** genome and tried our program on this genome with sequenced reads. However, with this dataset, we can't verify our program's performances in terms of results quality. We chose to analyse this dataset with **k_value = 20, h_value = 5 and min_abundance = 5**.

In the VCF file we obtained, we can observe a **significant difference in terms of abundances** between SNPs detected. We can see that there is only one SNPs with an abundance of 6 whereas the other abundances are between 14 and 68.

If we **decrease the abundance value** to 3, we find 12 new SNPs with abundances of 3 or 4. It is likely that these observations are due to sequencing errors but knowing the virus tendency to mutate, it can be interesting to keep them to observe if they are detected with other sequenced reads and can be the sign of a new mutant.

When we **increase the number of autorised mismatches** from 5 to 20, we find a modification of our results with 38 SNPs found instead of 19. Some of these observations are interesting because found in high abundance (almost 15) and can be signs of mutants.

Finally, to be rigorous in our research, we could check the **impact of these SNPs** on the virus behavior and see if these mutations have an impact on viral proteins or if they are silenced. When we look at Figure 1 presenting the SNPs observed with values k_value = 20, h_value = 5 and min_abundance = 5, we can see that substitutions observed are mainly **transitions** (*substitutions between nucleotids belonging to the same family*) between C and T. These are frequent substitutions and it is said that transitions are more likely to conserve biochemical properties of the amino acid than transversions and thus are **less detrimental**.

Some of the variants observed here are already present in Covid 19 Data Portal and seems frequent but others are not yet reported : they can be specific to the virus we sequenced here or not yet referenced.

Recently, a new variant of SARS-CoV-2 called VUI – 202012/01 appeared with **17 mutations** compared to the SARS-CoV-2 genome sequenced in Wuhan. It would be interesting to see if these

mutations were already present in the virus sequenced from patient id = SRX9435498. This could help to understand the evolution of the virus and its mutations.

```
#REF: covid/MN908947.fasta
#READS: covid/SRX9435498_subset10000.fasta
#K: 20
#MAX_SUBST: 5
#MIN_ABUNDANCE: 5
240 C   T   41
1058    C   T   61
3036    C   T   41
10137   C   T   50
10318   C   T   42
14407   C   T   65
16851   G   T   65
18423   A   G   41
21303   C   T   14
21810   C   T   57
23402   A   G   68
25562   G   T   52
25906   G   T   29
26106   G   T   29
27463   T   C   23
27963   C   T   15
28471   C   T   30
28868   C   T   19
29869   C   A   6
```

Figure 1 - Example of VCF file obtained during Covid data analysis