Rapport utilisateur

Cet outil permet de trouver et quantifier les mutations observées en mappant une séquence de référence avec plusieurs reads.

L'outil se décline en deux parties, l'indexation et le mapping.

Indexation

Le programme index.py permet d'indexer un génome de référence, fourni sous la forme d'un fichier fasta. L'indexation permet de créer un tableau d'informations (index) nécessaire pour le mapping.

Cet index est créé comme présenté sur la Figure 1. L'index permettra ensuite d'appliquer le principe de la *Burrows Wheeler Transform* (BWT), un algorithme de compression des données, qui permet de retrouver rapidement une sous-chaîne de caractère dans une séquence référence. Pour plus d'informations : Principe détaillé de l'algorithme BWT

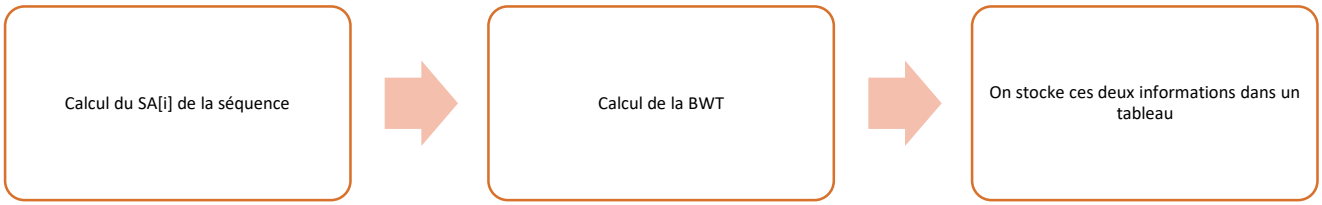

Figure 1 : Principe de l'indexation

L'index retourné est constitué de deux colonnes, la première correspondant au sa[i] (int) et la deuxième à la transformée de Burrows Wheeler (str) soit le dernier nucléotide de chaque séquences classées par ordre lexicographique.

Ce premier programme prend en entrée les arguments suivants:

```
python index.py --ref [genome_file.fa] --out [dumped_index.dp]
```

Avec :
ref le fichier contenant la séquence de référence, au format fasta,
out le fichier de sortie, au format dp.

Mapping

Le programme map.py permet d'aligner les reads avec la séquence de référence, de compter le nombre de substitutions à chaque position et retourner un fichier d'informations sur le mapping.

Le programme se base sur le principe d'alignement « seed and extend » :

L'idée est de trouver une « ancre » dans chaque read. Une ancre est une sous chaîne de caractère du read qui s'aligne parfaitement à une certaine position de la séquence référence, et à partir de laquelle l'alignement commencera. Ainsi pour chaque read, on cherche toutes les ancres existantes de taille *k*, définit par l'utilisateur. Pour chaque ancre, on cherche une correspondance parfaite (pas de substitutions) avec la séquence de référence, et ce à l'aide de l'algorithme de BWT (recherche d'un pattern dans une séquence de référence).

Une fois la position d'ancrage trouvée dans la séquence de référence, on étend l'alignement de chaque côté de l'ancre de manière à aligner complètement le read avec la séquence référence. Lors de cet alignement, on compte le nombre de substitutions entre le read et la séquence de référence. Ce nombre ne doit pas dépasser *max_hamming*, un nombre fixé par l'utilisateur. Pour chaque read, l'alignement avec le moins de substitutions est ensuite sélectionné. S'il y a des alignements ex-aequo, on choisit celui qui commence le plus à gauche dans la séquence de référence. On inscrit ensuite dans un fichier les substitutions observées (nucléotide originel, nucléotide variant) et le nombre de fois qu'on les observe. Les substitutions sélectionnées sont seuillées par *min_abundance*, une valeur indiquée par l'utilisateur, qui correspond au nombre minimum d'observation d'une substitution.

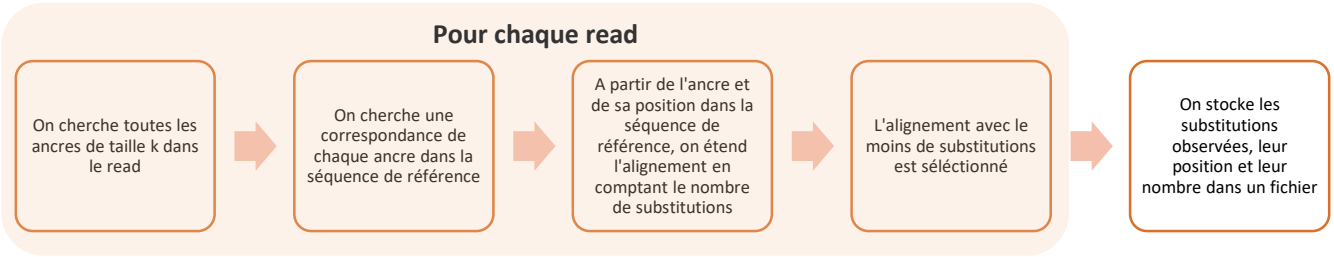

Figure 2 : Principe du mapping

Le fichier retourné est au format .vcf. Il se compose comme suit :

##REFERENCE : nom du fichier contenant la séquence référence

##READS : nom du fichier contenant les reads

##K : taille des ancres

##MAX_SUBS : nombre de substitutions maximum autorisé

##MIN_ABUNDANCE : nombre d'observations des substitutions minimum autorisé

##SUBSTITUTION INFORMATION

#POS REF ALT ABUNDANCE

Position de la substitution	Nucléotide de référence	Nuclétoide alternatif	Abondance de la substitution à cette position

Le mapping prend en entrée les arguments suivants :

```
python map.py --ref [genome_file.fa] --index [dumped_index.dp] –reads [reads.fa] -k [k_value]
--max_hamming [h_value] --min_abundance [m_value] –out [snps.vcf]
```

Avec :
ref le fichier contenant la séquence référence, au format fasta,
index le fichier index crée lors de l'indexation, au format dp, ,
reads le fichier contenant les reads, au format fasta,
k la taille des ancres,
max_hamming le nombre de substitutions maximum autorisé,
min_abundance le nombre d'observations des substitutions minimum autorisé,
out le fichier de sortie, au format vcf

Performances

Cette partie a pour but d'aiguiller sur le choix des paramètres et l'impact de leur variation sur les performances des résultats. Pour obtenir ces résultats, nous avons utilisé une séquence de référence de longueur 15 000 et 30 000 reads de longueur 100. Le paramètre max_hamming a été fixé à 5 (5% de mutations autorisé sur un read) et le paramètre min_abundance a été fixé à 10 (soit 5% d'observation de la mutation sur la profondeur moyenne).

k	Temps de calcul (s)	Mémoire utilisée (% CPU)	Qualité des résultats	
			Précision (%)	Rappel (%)
10	1163.227	15	98.2	99.7
15	997.59	14.9	98.2	99.7
20	1045.661	14.7	98.2	99.7
25	1076.76	17.8	98.1	99.7
30	1170.836	17.1	97.7	99.7
35	1001.217	14.1	95.6	99.9
40	1007.5	16.2	91.2	99.9
45	976.869	16.5	82.4	100
50	916.33	16.6	66.7	100
55	863.507	25	49.5	100
60	827.874	29.8	30.8	100
65	719.553	15.9	16.2	100
70	622.725	14.9	4.8	100

Tableau 1 : Performances en faisant varier k et en fixant max_hamming = 5 et min_abundance = 10

La précision décline de plus en plus vite en augmentant la taille des ancres, le rappel quant à lui n'est pas réellement impacté par la taille des ancres. On observe également un « minimum local » du temps de calcul pour k = 15, entre des tailles allant de 10 et 30. Après 30, l'augmentation des tailles des ancres entraine une

diminution du temps d'exécution. Enfin, la mémoire utilisée varie peu, sauf pour des tailles d'ancre de 55 à 60, pour lesquelles il augmente beaucoup. Pour déterminer le meilleur paramètre de k, on cherche à optimiser le couple Précision/Rappel, puis dans un second temps le temps d'exécution et la mémoire utilisée. Nous recommandons une taille d'ancres entre 15 et 20 dans ce cas (soit 15 à 20% de la taille des reads), avec de bons résultats (précision = 98.2%, rappel = 99.7%), pour un temps d'exécution le plus faible possible. Le temps de calcul est relativement lent pour ce genre de programme, pour pouvoir optimiser les autres paramètres nous continuerons avec une taille d'ancre de 15. Cela est dû à l'utilisation de dictionnaires pour stocker des variables, qui ralentit l'exécution de l'algorithme.

max_hamming	**Temps de calcul (s)**	**Mémoire utilisée (% CPU)**	**Qualité des résultats**	
			Précision (%)	**Rappel (%)**
1	981.157	14	5.7	100
3	990.099	14	83.2	99.9
5	1002.962	15.2	98.2	99.7
7	1002.72	17.3	98.7	99.7
9	1070.465	14	98.7	99.7
11	994.951	13.7	98.7	99.6
13	971.305	15.5	98.7	98.7
15	964.971	17.9	98.7	98.7

Tableau 2 : Performances en faisant varier max_hamming et en fixant k = 15 et min_abundance = 10

On remarque que la précision commence à se stabiliser à partir de 7 et plus, pour une très bonne précision (98.7%). Le temps de calcul augmente pour un paramètre *max_hamming* allant de 1 à 9, puis diminue au-delà de 9. Le rappel décline très légèrement quand on augmente le paramètre. Enfin, la variation de la mémoire semble aléatoire.

Ainsi, nous recommandons de fixer ce paramètre entre 7 et 11 dans ce cas (soit entre environ 5 à 10% de la taille d'un read). Si le paramètre *max_hamming* est trop bas, cela entraînera une chute de la précision. A l'inverse, s'il est trop élevé, cela entraînera une perte de rappel. Pour la suite, nous fixerons le paramètre *max_hamming* à 7.

min_abundance	Temps de calcul (s)	Mémoire utilisée (% CPU)	Qualité des résultats	
			Précision (%)	Rappel (%)
2	1015.105	20.4	99.6	51.3
4	982.891	17.9	99.4	98.6
6	979.067	15.7	99.4	99.3
8	1018.565	17.8	99.3	99.7
10	985.544	15.9	98.7	99.7
12	1027.32	21.3	95.5	99.8
14	1353.927	20.2	90.5	99.8
16	979.63	13.9	79.5	99.9
18	1024.972	20.7	63.7	99.8
20	1001.098	16.2	46.8	99.8

Tableau 3 : Performances en faisant varier min_abundance et en fixant k = 15 et max_hamming = 7

La précision, très élevée (> 99 %) pour de faibles valeurs du paramètre *min_abundance (< 8)*, décroît de plus en plus rapidement quand le paramètre augmente. Le rappel, quant à lui, est faible (51.3 %) pour de petites valeurs de *min_abundance (< 3)* et augmente très rapidement quand le paramètre augmente avant de se stabiliser pour *min_abundance* = 8 (> 99.7 % de précision). En revanche, il semble que le temps de calcul et la mémoire utilisée ne soient pas corrélés avec le paramètre min_abundance.

Nous choisissons de fixer ici le paramètre *min_abundance* à 6 (soit 3% de notre profondeur moyenne) pour avoir une très bonne précision et un très bon rappel (> 99 %), et un temps d'exécution relativement faible. Les meilleurs résultats, sont, d'après ces tests, obtenus pour des valeurs du paramètre *min_abundance* comprises entre 2 et 4% de la profondeur moyenne.

Analyses des données *covid*

Les données du COVID ont pu être analysées grâce aux paramètres optimisés sur le jeu de validation d'Escherichia coli, soit : taille des kmer : 15 ; maximum de substitutions autorisées par reads : 7 ; minimum de répétition des substitutions par localisation : 6.

On se rend compte suite à l'analyse que de nombreuses localisations sont centralisées sur le génome (ex : 10137-10318 et 21303-21810-23402 et 25562-25906-26106) ce qui montre que de nombreuses portions du génome sont conservées. Cela n'est pas aberrant puisqu'il s'agit d'un coronavirus, une famille de virus déjà connue et avec une structure stable. La différence dans ce virus COV-SARS2 est les peptides présents sur la capside virale et les particules virales. Grâce à la carte du génome viral il est possible de voir qu'il est composé de 6 ORFs (les ORFs étant les sections virales transmises à l'hôte lors de l'infection) ; sur ces 6 ORFs, 4 possèdent des mutations recensées dans le fichier vcf. De ce fait, la majorité des protéines vont être touchées dû à ces mutations puisque les ORFs coderont les protéines chez l'hôte. De plus, certaines mutations sont localisées directement sur des séquences protéiques ce qui peut modifier du tout au tout la protéine finale et donc l'impact du virus sur l'hôte. Les différentes protéines touchées sont variées et n'ont pas le même rôle. Il est donc difficile de faire un lien entre ces protéines et d'apporter une réponse sur les mutations détectées.

Pour conclure, les différentes mutations trouvées touchent soit des ORFs sans toucher de zones protéiques ou des parties d'ORF directement liées à une protéine en particulier. Ces différentes mutations ont donc un impact non sans conséquences sur la réplication des protéines et de ce fait la virulence du virus chez l'hôte.

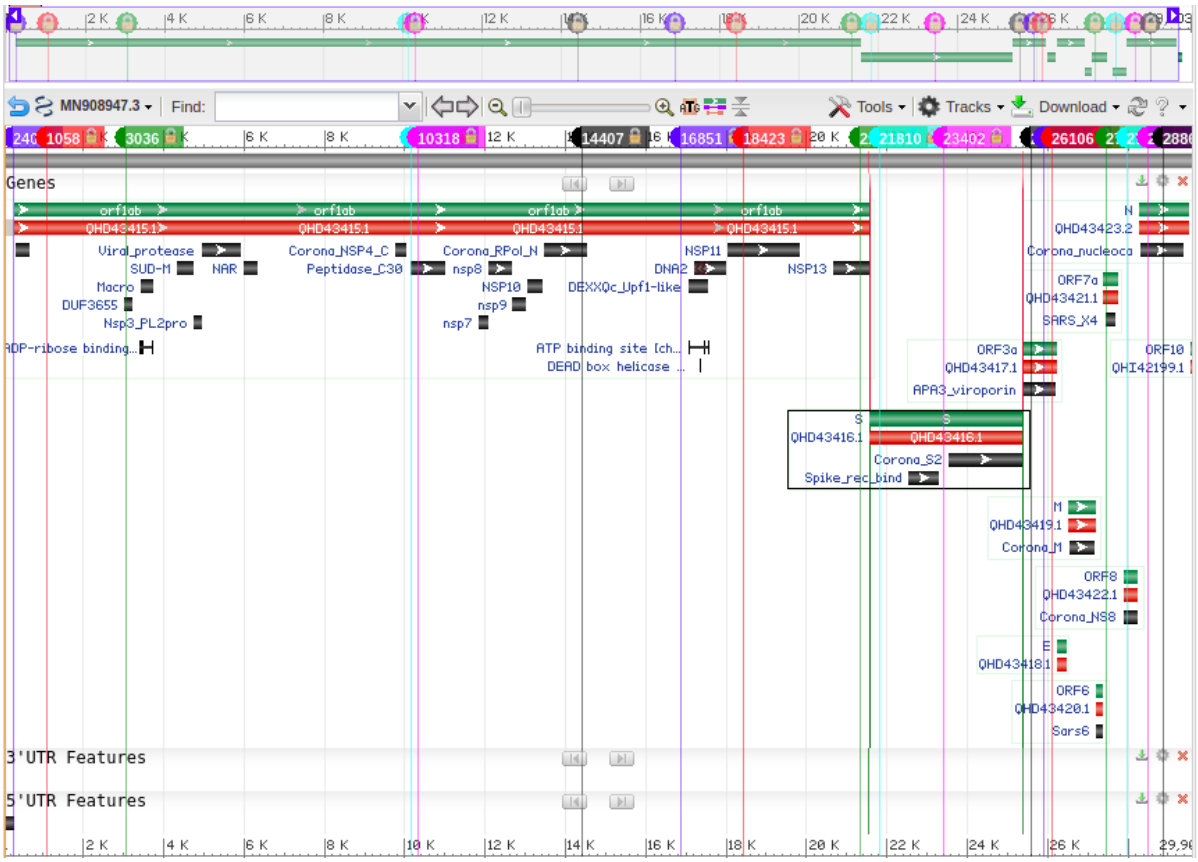

Figure 3 : Mapping du génome complet du SARS COV2 obtenu sur NCBI (https://www.ncbi.nlm.nih.gov/nuccore/MN908947.3?report=graph), chaque étiquettes colorées correspondent aux localisation des mutations détectées sur le fichier de sortie vcf