Bio-informatique

Rapport Utilisateur – Thomas Esseul – IL1.1

Présentation

Contexte d'utilisation

Les fichiers fournis ici servent à compresser un fichier au format fasta (détail du format utilisé dans la partie suivante). Pour cela, ils utilisent le principe du Graphe de De Bruijn représenté par un Filtre de Bloom. Nous allons ici parler de son utilisation.

Rappels sur le format fasta

Un fichier fasta doit utiliser l'extension *.fasta*, il représente des séquences de nucléotides. Dans notre cas, il contient un nombre donné de reads, tous composés exclusivement des caractères A, C, G et T. En plus des reads, le fichier peut contenir des annotations, auquel cas les lignes de ces annotations doivent commencer par le caractère ">". Par exemple :

> nucl seq 5986 read 5
TACGCCAATCAGCGCGTGCAGTTTGGCGAGGCTATT
> nucl seq 5986 read 6
GGGAACGCGCGGGCTTTTTCTAACGCCAGCGCCGC

Utilisation

Les fichiers de compression et de décompression (respectivement *Compressor.py* et *Decompressor.py*) sont utilisables via des lignes de commandes, incluant des paramètres positionnels (obligatoires) et optionnels à utiliser comme suit. L'impact des paramètres sur les performances du compresseur seront discutées dans une prochaine partie.

Compressor.py

L'utilisation du fichier de compression se fait via la commande :

python Compressor.py fasta [options]

où :

- *fasta* indique le chemin du fichier .fasta à compresser
- options peut inclure un nombre indéfini de paramètre
    - --graph *graphPath* : Indique le chemin du fichier de sortie au format .graph.pgz contenant le graph de De Bruijn
    - --output *outputPath* : Indique le chemin du fichier de sortie au format .comp contenant le contenu du fichier fasta compressé
    - --kmer_size *size* : Un entier indiquant la taille de kmer à utiliser (qui doit être plus petite que ou égale à la taille du plus petit read du fichier fasta)
    - --bloom_size *size* : Un entier indiquant la taille du filtre de Bloom à utiliser
    - --bloom_hash *hashNb* : Un entier indiquant le nombre de fonction de hash à utiliser

Tous les paramètres optionnels ont des valeurs par défaut, notamment *graph* et *outputPath* qui, s'il ne sont pas précisés par l'appel du fichier, seront générés automatiquement à partir du nom du fichier d'entrée.

Decompressor.py

L'utilisation du fichier de décompression se fait via la commande :

python Decompressor.py comp [options]

où :

- *comp* indique le chemin du fichier .comp à décompresser
- options peut inclure un nombre indéfini de paramètre
    - --graph *graphPath* : Indique le chemin du fichier d'entrée en format .graph.pgz contenant le graph de De Bruijn
    - --output *output* : Indique le chemin du fichier de sortie au format .fasta contenant le contenu du fichier fasta décompressé

A l'image du compresseur, les paramètres optionnels indiquant des chemins de fichiers, s'il ne sont pas précisés, seront générés automatiquement à partir du nom du fichier d'entrée. Ainsi si l'utilisateur réalise l'appel en précisant uniquement le fichier "read10x.comp", le programme cherchera un fichier "read10x.graph.pgz" dans le même répertoire.

Performances

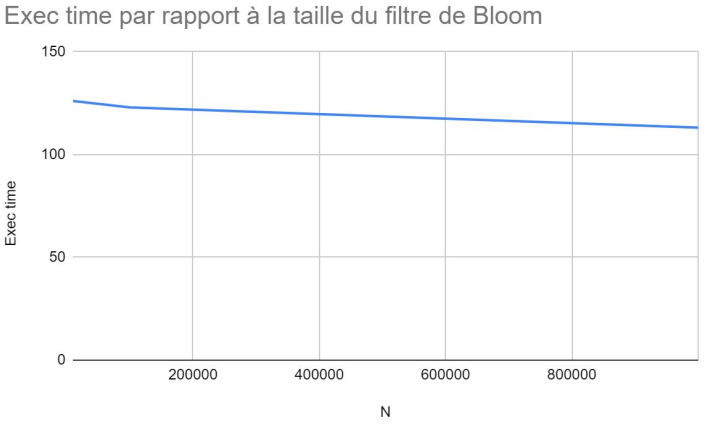
On se propose maintenant de commenter les performances du programme dans sa globalité, les résultats qui suivent correspondent à l'exécution d'une compression suivi d'une décompression sur un fichier contenant 15.000 reads, pour un poids total de 1.683 Ko.

Pour réaliser ces comparaisons, on fait varier trois paramètres, à savoir la taille du filtre de Bloom, le nombre de fonctions de hash et la taille des kmer considérés; et on observe la taille des fichiers (en Ko) de sortie ainsi que le temps d'exécution. Une fois un nombre de données suffisant récolté, on sera en mesure de faire des moyennes sur les gains et perte de temps et d'espace.

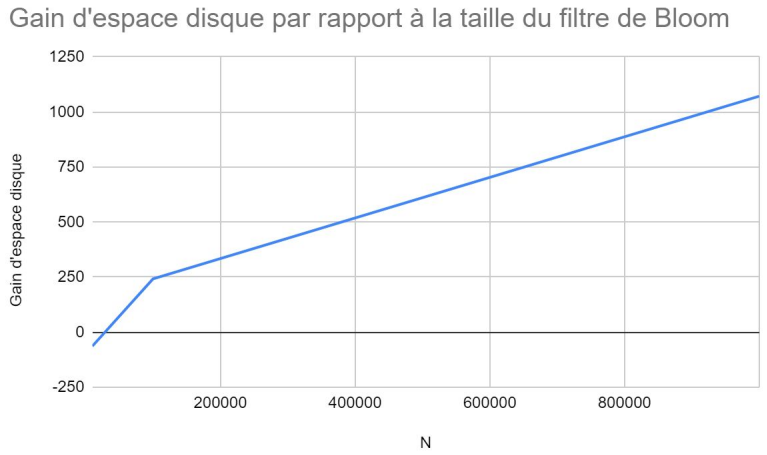
Pour les trois paramètres, on calcule le "Gain d'espace disque" obtenu par la formule:

gain = TailleFichierOriginale - (TailleFichierCompressé + TailleGrapheCompressé)

Taille de filtre de Bloom



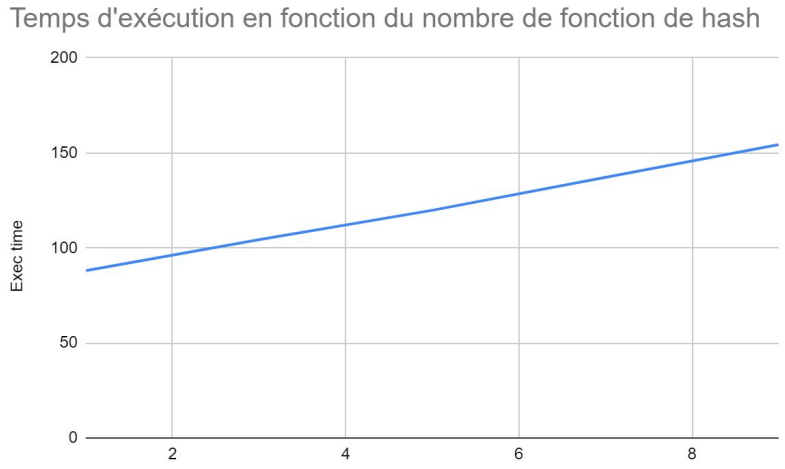
On remarque ici que la taille du filtre de Bloom n'a qu'un impact mineur sur le temps d'exécution, le gain n'étant pas extrêmement grand, on pourrait choisir de le négliger devant d'autres paramètres plus impactants.



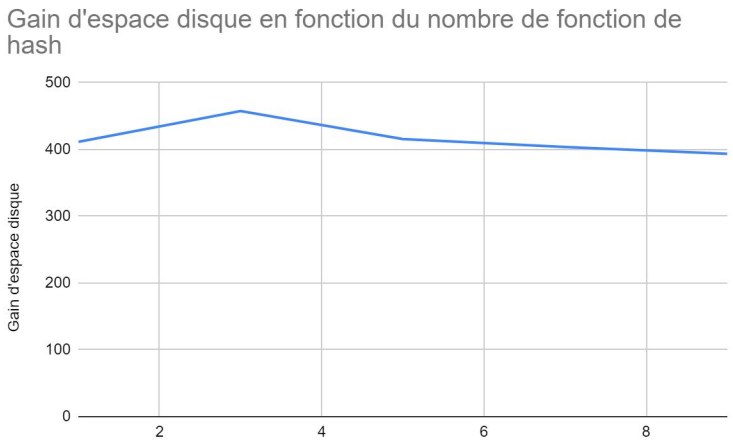
En revanche, on remarque un clair gain d'espace disque accompagnant la croissance de cette taille. Néanmoins il ne faut pas oublier que faire grandir le filtre représente aussi une augmentation de la mémoire vive mobilisée pendant l'exécution du programme, l'augmentation doit donc rester raisonnable.

On conclura qu'une taille conséquente du filtre ne soit pas à craindre, en revanche il va de soit qu'elle se doit d'être proportionnelle à la taille du fichier d'entrée.

Nombre de fonctions de hash



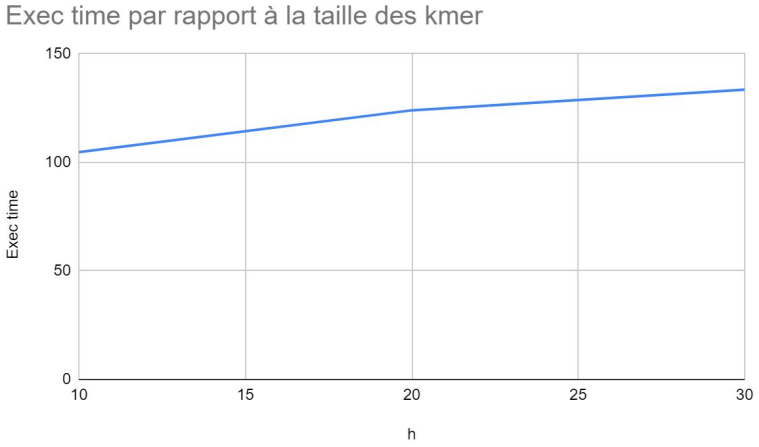
Sans surprise, on constate que le nombre de fonction de hash a un impact sur le temps d'exécution d'autant plus important qu'il est grand, cela s'explique très simplement par le fait que plus de fonctions utilisées demandent plus de temps de calcul.



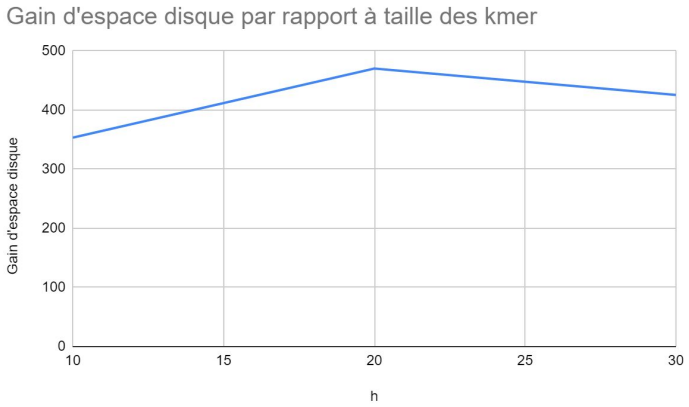
Pour ce qui est de l'espace disque, on constate une tendance inverse, la croissance de ce nombre semble représenter un gain d'espace, quoiqu'assez minime.

On peut en conclure qu'il est conseillé d'utiliser au moins 5 fonctions de hash, ce qui permet de réduire l'espace disque utilisé de façon satisfaisante sans augmenter le temps d'exécution de façon trop conséquente.

Taille des kmers



Pour ce qui est du temps d'exécution, on constate que plus les kmers considérés sont grands, plus le temps d'exécution augmente, une tendance pouvant sans doute être expliquée par le fait que de plus grand kmer demandent plus de calcul lors du hashage de ceux-ci.



Enfin, le gain d'espace disque, lui, suit une courbe surprenante puisque le pic de gain se trouve sur la valeur précise de 20, le phénomène ne semblant pas avoir d'explication évidente, on se contentera de supposer cette valeur est celle de l'équilibre entre taille du fichier compressé et taille du filtre de Bloom compressé.

On retiendra donc que la taille des kmers utilisés doit être adapté en fonction du rapport recherché entre espace disque utilisé et temps d'exécution nécessaire.

Conclusion

Pour conclure sur les variations des différents paramètres et les gains et pertes que celles-ci entraînent : Pour ce qui des fonctions de hash, un minimum semble être requis néanmoins l'intérêt de monter au delà de ce minimum semble limité. En ce qui concerne la taille des kmers, en l'absence d'études plus approfondies, on se contentera de la valeur d'équilibre mise en évidence ici. Enfin, on recommandera de privilégier l'utilisation d'un filtre de Bloom conséquent puisqu'il est, des trois paramètres étudiés, celui qui apporte le plus de gain (temps et espace) par sa croissance.