Assemblage de Génome De Novo
Rapport de Projet Court

Etienne JEAN

Septembre 2018

1 Introduction

L'apparition récente des techniques de séquençage à haut débit a ouvert la voie à de nouvelles possibilités dans le domaine de la génomique. Ces techniques génèrent des lectures courtes de séquences (appelées dans la suite de ce rapport "short reads") en très grand nombre, et pour des couts faibles. Ainsi, il devient possible d'étudier le génome de nombreux organismes par séquençage. Le problème est que dans un premier temps, ceux-ci n'ont pas de génome de référence. Il convient donc de l'assembler à partir des shorts reads générés par séquençage à haut débit. Ce projet avait pour but d'implémenter un assembleur de petits génomes à un chromosome circulaire.

Deux reads qui se superposent sur une partie de leur longueur recréent une séquence plus longue. Si cette superposition ne se retrouve qu'une fois dans le génome, alors l'association des deux reads forme ce qu'on appelle un contig, qui est inclus dans le génome de départ. En répétant ce processus, on peut espérer ré-assembler le génome qui a été séquencé. Etant donné qu'il est impossible de déterminer à priori si une superposition est unique dans le génome, on ne considère que les superposition d'une certaine longueur, au-delà de laquelle la probabilité d'unicité est très proche de 1.

En pratique, il convient de découper les reads en morceaux de longueurs k, appelés k-mers. L'objectif est d'obtenir autant de k-mers différents qu'il y a de nucléotides dans le génome. Ainsi, ces k-mers pourront etre assemblés avec des superposition de (k-1)-mers afin de recréer le génome.

De là, il existe deux stratégies. Soit les k-mers constituent les sommets d'un graphe, et ils sont reliés par les superposition de (k-1)-mers en tant qu'arêtes, soit, à l'inverse, les (k-1)-mers forment les sommets, et les k-mers constituent les arêtes (Graphe de De Bruijn). Dans les deux cas, ces graphes sont orientés. Dans la première représentation, il convient de trouver un cycle, dit Hamiltonien, qui passera par tous les sommets exactement 1 fois. Dans la seconde, il faut trouver un cycle, dit Eulerien, qui passera par toutes les arêtes exactement une fois, pour reconstituer le génome.

La recherche de cycles Hamiltonien est un problème d'algorithmique NP-complet, mais pas la recherche de cycles Eulérien. Ainsi, les graphes de De Bruijn sont particulièrement adaptés aux nouvelles technologies de séquencage a haut débit, qui produisent un très grand nombre de petits reads.

2 Démarche & Justification des choix

2.1 Description des modules

Le code est décomposé en 4 modules.

— Module `genome_generation.py` : Ce module permet de générer un génome aléatoirement, d'une longueur `l` et avec un pourcentage de GC `g`. Ce module est optionnel, pour pouvoir utiliser le programme avec un genome existant.

— Module `sequencing.py` : Ce module permet de simuler le séquençage à haut débit d'un génome circulaire. Il prend un genome au format fasta, la taille des reads et la couverture de séquençage

(profondeur de séquençage) en paramètres d'entrée, et renvoie un fichier de reads, pris a des positions aléatoires du génome.
— Module `denovo_assembly.py` : Ce module est le corps du projet. Il prend en parametres un fichier de reads, et essaye de les assembler pour recréer le génome de départ. Pour ce projet, j'ai donc fait le choix d'implémenter un graphe de De Bruijn, et d'y rechercher les cycles d'Euler.
— Module `test_assembly.py` : Ce module prend en paramètres le génome de référence et le génome réassemblé, et teste l'égalité entre les deux.

2.2 Justification des choix

Parmi les deux implémentations des graphes, listes d'adjacence et matrices d'adjacence, j'ai choisi la liste d'adjacence. Les matrices d'adjacence sont meilleures pour les graphes denses, avec de nombreuses arêtes, mais prennent beaucoup de mémoire inutilement dans le cas de graphes peu denses, comme c'est le cas avec un graphe de Bruijn pour l'assemblage de génome.

L'implémentation du graphe a été totalement faite en programmation orienté objet. Elle est composée de 3 classes.
— Classe `Edge` : Cette classe représente les arêtes du graphe, et chaque arête a un pointeur vers les sommets de départ et d'arrivée.
— Classe `Vertex` : Cette classe représente les sommets du graphe, et de même chaque sommets a également un pointeur vers les arêtes entrantes et les arêtes sortantes.
— Classe `Graph` : Cette classe regroupe une liste d'objets Edge, une liste d'objets Vertex, et l'ensemble des méthodes qui permettent de générer le graphe, de trouver les cycles d'euler, de les assembler et de recréer la séquence initiale.

Ajouter les arêtes entrantes et sortantes au objets sommets n'est pas indispensable, mais en les gardant en mémoire, il permet de tester si le graphe contient un cycle Eulérien passant par toutes les arêtes et il permet de faire la recherche de cycles Eulérien de manière beaucoup plus efficace. Le temps d'exécution a été divisé par 20 de cette manière.

Le programme gère les reads de taille variable. Lee séquençage en réalité n'étant pas parfait, il arrive que les reads ne soient pas tous de la même longueur.

2.3 Difficultés rencontrées

J'ai eu beaucoup de difficultés à trouver une bonne implémentation du graphe en programmation orienté objet, et c'est ce qui m'a pris le plus de temps. L'algorithme d'assemblage des cycles n'était également pas évident à trouver. Enfin, il a été assez difficile de trouver une implémentation relativement optimisée, qui ne garde en mémoire que le nécessaire (pas les reads par exemple), et soit efficace à utiliser.

3 Résultats

3.1 Tests du programme

Le programme semble assez efficace. J'ai pu assembler en 1.2s un genome de 10kb, séquencé avec des reads de 100pb, une couverture de 20x (20 fois la taille du génome est séquencée), et des k-mers de 30pb.

Pour un génome aléatoire plus grand, de 1Mb, l'assemblage s'est effectué en 92.1s, avec des reads de 100pb, une couverture de 50x, et des k-mers de 55pb.

Enfin, j'ai récupéré la séquence du génome de Mycoplasma genitalium sur GenBank (code NC_000908.2), et j'ai tenter de la ré-assembler. Je n'ai reussi a effectuer un assemblage complet que pour des paramètres assez irréalistes : Le génome fait 586kp, l'assemblage s'est effectué en 83.1s avec des reads de 400pb, une couverture de 100x, et des k-mers de 250pb. Cela suggère qu'il y a des petites séquences

répétées de quelques centaines de paires de bases dans le génome de Mycoplasma genitalium, qui empeche l'établissement d'un graphe de De Bruijn équilibré (sans cycle d'Euler qui passerait par toutes les arêtes), pour des valeurs de k-mers réalistes.

3.2 Ajustement des paramètres

En théorie, tout genome à 1 chromosome circulaire peut être assemblé avec ce programme, si les reads sont suffisament longs et suffisamment nombreux. Pour pouvoir assembler entièrement, il faut et il suffit d'avoir autant de k-mers différents qu'il y a de paires de bases dans le génome.

Ainsi,

— Plus les kmers sont courts, plus il y a de chance qu'un k-mer se retrouve à 2 positions dans le génome. Solution : agrandir la taille des k-mers

— Cependant, plus les k-mers sont de taille proche de celle des reads, plus il y a de chances de ne pas trouver un k-mer à chaque position du genome. Solution : augmenter la couverture, ce qui est réaliste avec la technologie de séquençage actuelle.

— Enfin, si les k-mers ne sont toujours pas suffisament longs, il faut alors augmenter la taille des reads, mais des reads de plus de 200pb deviennent irréalistes avec les techniques de séquençage à haut débit.

Le plus souvent, lorsque l'assemblage réussit, il n'y a qu'un seul cycle dans le graphe. Cela indique que les k-mers sont longs. Il existe une longueur de k-mer au-dessus de laquelle, le graphe est composé d'un seul cycle, car tous les k-mers, et tous les (k-1)-mers sont uniques. En-dessous de cette valeur cependant, dans la majorité des cas, le graphe n'est plus équilibré car les k-mers, ne sont plus uniques. Enfin, il existe un intervalle critique de longueurs de k-mers, pour lesquelles le graphe est composé de plusieurs cycles. C'est le cas quand les k-mers sont uniques, mais pas les (k-1)-mers, cependant le programme peut quand même assembler ces cycles pour recréer un génome. Leur taille de diffèrent que de 1pb, donc cet intervalle est souvent très étroit. Pour un génome de 10kb par exemple, à k=12, l'assemblage n'est pas possible, à k=13 il y a 2 cycles, et à k=14, il n'y a déja plus qu'un cycle.

4 Améliorations

Le programme ne permet pas de gérer certains problèmes, et mériterait des améliorations.

— Erreurs de séquençage : Ce code ne gère pas les erreurs de séquençage. Il n'est donc pas capable d'assembler de vrais données de NGS. J'aurai bien aimé essayer de gérer ce problème, mais le temps m'a manqué. C'est pourquoi le module `sequencing.py` n'est pas optionnel, et doit nécessairement être utilisé pour générer des reads parfaits.

— Assemblage multiples : Il existe des cas ou les assemblages peuvent etre multiples (quand 2 cycles ont en commun 2 sommets différents). En pratique c'est assez rare. Ce code permet de trouver 1 assemblage, mais ce ne sera pas forcément le bon. Il aurait été bien d'avoir une fonction pour renvoyer tous les assemblages possibles.

— Assemblage en contigs. Il aurait été bien de permettre de renvoyer des cycles d'Euler présents dans un graphe non entièrement équilibré (quand il y a des séquences répétées). On renvoit ainsi une liste de contigs, qui couvrent la majeur partie du génome. Ceux-ci pourraient même être assemblés entre eux grâce aux techniques de séquençage "long reads" (genre Nanopore), qui couvriraient les longues séquences répétées.

— Assemblage de chromosome linéaire : Avec seulement quelques modifications mineures, ce code pourrait gérer l'assemblage de chomosomes linéaire.

— Assemblage de génome à plusieurs chromosomes linéaires : En combinant les deux précédentes fonctionnalités, on peut imaginer assembler des genomes linéaires à plusieurs chromosomes, chaque contig étant un chromosome.