Rapport du projet court:

Assemblage de novo d'un génome circulaire

Auteur: Bellaïche Adam email: adam.bellaiche@gmail.com Université: Paris 7 Cursus: M2BI	Sujet de: Charlotte Périn email: charlotte.perin@univ-paris-diderot.fr

Année Universitaire: 2018-2019

1. Introduction

Le but de ce projet était de réaliser un programme permettant l'assemblage *de novo* d'un petit génome circulaire en utilisant un graphe de De Bruijn et en utilisant une des deux techniques présentées dans l'article:

Compeau, P. E., Pevzner, P. A., & Tesler, G. (2011). How to apply de Bruijn graphs to genome assembly. Nature biotechnology,29(11), 987.

Il y avait le choix entre:

- trouver un cycle hamiltonien: cycle dans un graphe passant par tous les sommets et une seule fois.
- trouver un cycle d'Euler: cycle dans un graphe passant par toutes les arêtes et une seule fois.

Ici, la technique d'Euler a été implémentée. En effet, il est moins coûteux en termes de ressources de trouver un cycle d'Euler qu'un cycle hamiltonien. Comme dit dans l'article, la charge de calcul qu'implique la recherche d'un cycle hamiltonien est si grande que la plupart des projets d'assemblages de génome ont abandonné cette stratégie car étant un problème NP-complexe, elle ne permet pas de travailler sur un nombre de read important comme par exemple 10^6 reads. Il était donc plus judicieux d'utiliser la stratégie d'Euler.

Pour rappel, un graphe de De Bruijn est un graphe orienté qui permet de représenter des mots de longueur *n* par des chevauchements de mots *n-1* construits à partir d'un alphabet donné. Ici, les mots de longueur *n* sont appelés des k-mers et les mots de longueur *n-1* les suffixes et les préfixes d'un k-mer. C'est à dire que le suffixe est le k-mer sans sa première lettre et le préfixe est le k-mer sans sa dernière lettre.

2. Matériel et méthodes

a. Matériel

Le programme a été codé en python 3. Il faut donc le lancer via une commande faisant appel à python 3 et non python 2. Ensuite, le programme est constitué de trois fichiers(cf: Annexe 1):

- main.py: qui effectue les taches principales en s'aidant des deux autres fichiers,
- Genome.py: qui permet de créer un génome soit aléatoirement soit à partir d'un fichier fasta. Cette classe permet aussi de créer les reads, les k-mers, et de vérifier si une séquence d'ADN est égale à une autre séquence d'ADN en considérant la circularité du génome.
- Graphe.py: qui permet de gérer un graphe. Cette classe est orientée de façon à respecter un graphe de De Bruijn, par exemple, elle n'admet pas des noeuds de même valeur. Elle sert aussi à créer une arête, ou à savoir si le graphe est Eulérien et à trouver un cycle d'Euler.

Il y a quatre bibliothèques extérieures utilisées: random, time, sys, copy.

Le code suit une pratique d'écriture, les variables sont écrites comme suit: nom_de_la_variable. Alors que les fonctions sont écrites comme suit: nomDeLaFonction. Chaque fichier a une docstring et une fonction help renvoyant cette docstring.

b. Méthodes

i. Générer les données

Le génome est construit soit aléatoirement soit à partir d'un fichier fasta. Les reads sont construits aléatoirement à partir du génome. Les k-mers sont construits à partir des reads, on vérifie préalablement qu'il n'y a pas deux fois le même k-mer. Les suffixes et les préfixes sont construits à partir des k-mers. Ils sont un k-mer moins sa première ou dernière lettre.

ii. Générer le graphe

Le graphe lui est généré à partir de l'ensemble des suffixes et préfixes distincts, d'où la vérification au préalable d'avoir des k-mers distincts car en ayant des k-mers distincts il n'y aura pas deux fois la même arête . En effet chaque noeud devient un des préfixes ou suffixes, et une arête est construite du préfixe vers le suffixe si ces deux se complètent.

iii. Assembler le cycle

Après avoir vérifié que le graphe obtenu est Eulérien (s'il ne l'est pas, il faudra relancer le programme), il faut maintenant trouver un cycle d'Euler. Pour cela, un algorithme est utilisé(cf: Annexe 2):

Un cycle 1 est formé en parcourant aléatoirement le graphe, en passant par les arêtes non empruntées et en partant d'un noeud quelconque tant que toutes les arêtes ne sont pas utilisées.

Tant que toutes les arêtes ne sont pas utilisées et si on revient au noeud de départ:

On prend un nouveau noeud dans le cycle 1 qui a toujours une arête non utilisée et on refait le cycle 1 jusqu'à arriver à ce nouveau noeud. En faisant ça, on aura trouver un cycle 2, le cycle 1 devient le cycle 2. On recommence le parcours aléatoire comme dit plus haut en partant du nouveau noeud.

On retournera le cycle passant par toutes les arêtes une seule fois. Ce sera un cycle d'Euler. En le parcourant, il sera possible de retrouver une séquence en concaténant la valeur des arêtes dans l'ordre. Une arête a pour valeur la concaténation des deux noeuds qu'elle connecte.

iv. Tests

La séquence retrouvée sera ensuite comparée avec celle du génome en sachant que le génome est circulaire. Si elle se superpose bien avec le génome alors le programme l'écrira dans un fichier fasta sinon il faudra relancer le programme avec des paramètres différents.

3. Résultats

Ici, plusieurs commandes ont été lancées:

Génome en pb	**nucléotides disponible**	**nombre de read**	**taille des reads**	**tailles des k-mers**	**temps d'exécution en seconde**	**séquence trouvée?**
10000	"ATCG"	5000	100	20	0.5	oui
100000	"ACTG"	50000	100	55	20	oui
500000	"ATCG"	20000	100	55	437	oui
mycoplasma genitalium (environ 600000)	"ACTG"	20000	400	300	538	oui

Tableau1: les commandes lancées, avec leurs paramètres, leur temps d'exécution et le résultat.

4. Conclusion

Le programme fonctionne correctement et trouve généralement une séquence, même si il aurait pu fonctionner plus rapidement.

Bien sur, il ne fait pas tout pour l'utilisateur. En effet, l'utilisateur doit bien ajuster ses paramètres en fonction du génome (alors qu'il est aléatoire ou connu), car si il existe quelques répétitions dans le génome par exemple, il faudra fortement augmenté la taille des k-mers ou des reads comme pour le mycoplasma genitalium.

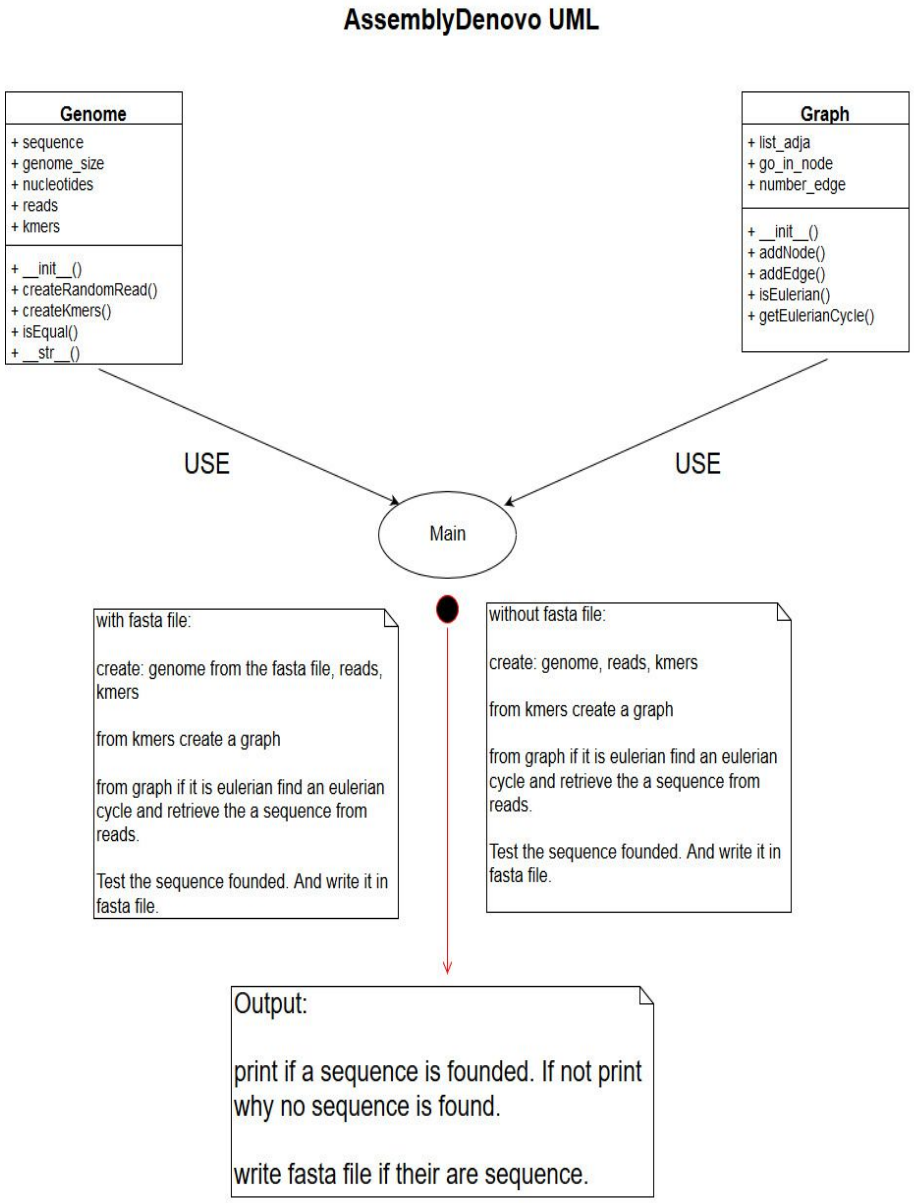
De plus, l'utilisateur devra parfois relancer le programme car il s'arrête si:

- La graphe construit n'est pas Eulérien.
- La séquence obtenu n'est pas similaire à la séquence d'origine.

L'article ne fournit pas de résultats, donc il n'y pas de comparaison possible.

ANNEXES

Annexe 1: Diagramme UML du programme



Annexe 2: Algorithme d'Euler tiré de la chaîne youtube: Bioinformatics Algorithms: An Active Learning Approach.

```
EulerianCycle(BalancedGraph)
  form a Cycle by randomly walking in BalancedGraph (avoiding already visited edges)
    while Cycle is not Eulerian
      select a node newStart  in Cycle with still unexplored outgoing edges
      form a Cycle' by traversing Cycle from newStart and randomly walking afterwards
      Cycle ← Cycle'
  return Cycle
```