Master Sciences et Numérique pour la Santé
Parcours Bioinformatique, Connaissances, Données

HMSN307 : Rapport bibliographique

Outils d'assemblage de génomes

Anaïs LOUIS

Encadré par :
Sèverine BÉRARD

Sous la tutelle de :
Alban MANCHERON

Octobre - Décembre 2019

Table des matières

1. Introduction

Depuis l'apparition des nouvelles technologies de séquençage (aussi connues sous le nom de NGS) dans les années 2000, le nombre de données générées, appelées *reads*, à considérablement augmenté pour un coût de séquençage de plus en plus faible. L'un des enjeux de cette avancée technologique est l'assemblage de génome. En effet, avant l'arrivée des NGS séquencer un génome coûtait très cher et prenait beaucoup de temps, c'est pourquoi seul un faible nombre de génomes de petite taille était séquencé.

L'assemblage d'un génome consiste à reconstruire une séquence initiale à partir des *reads* obtenus par le séquençage de cette même séquence. On distingue deux types d'assemblage :

- L'assemblage avec génome de référence ((A) annexe 5.1) : les *reads* sont tout d'abord alignés à un génome de référence puis sont assemblés en une unique séquence.
- L'assemblage *de novo* ((B) annexe 5.1) : les *reads* sont dans un premier temps assemblés en contigs. Ensuite, les contigs sont assemblés en scaffolds puis les scaffolds sont assemblés pour former la séquence finale.

En parallèle aux améliorations des technologies de séquençage, de nombreux outils d'assemblage ont vu le jour. Parmi ces outils, trois grands types d'algorithmes utilisés pour assembler les génomes se distinguent :

- les algorithmes gloutons
- les algorithmes *Overlap Layout Consensus*
- les algorithmes basés sur le graphe de De Bruijn

Dans le cadre de ce projet bibliographique, l'intérêt s'est uniquement porté sur l'assemblage de génome *de novo*. Pour commencer, une première partie abordera les différents algorithmes utilisés par les outils d'assemblage de manière à avoir une vue globale de leur fonctionnement. Ensuite, une seconde partie se focalisera sur les outils d'assemblage afin d'avoir une vue d'ensemble de ce qui existe à ce jour.

2. Algorithmes d'assemblage de génome

Lorsqu'on s'intéresse aux outils d'assemblage existant, l'une des questions que l'on peut se poser est de savoir quel algorithme est utilisé. Ainsi, l'objectif de cette partie sera de décrire de manière succincte les différents algorithmes implémentés par les outils afin d'avoir une idée globale de leur fonctionnement. Trois méthodes se distinguent : la méthode gloutonne, la méthode OLC et la méthode du chemin Eulérien. [1]

2.1 Méthode gloutonne

La méthode gloutonne est un méthode simple et intuitive qui se base sur le calcul d'un score de chevauchement pour assembler les reads en contigs. Ce score de chevauchement sera calculé pour chaque paire de reads, et la paire qui aura le score le plus élevé sera fusionnée en premier. On continu ainsi de suite jusqu'à ce que toutes les séquences soient associées à une autre. (Figure 2.1)

Ce type d'algorithme est très gourmand en mémoire et à des difficultés à placer correctement les régions répétées. En effet, deux régions répétées seront généralement assemblées en contig du fait de leur score de chevauchement élevé alors que dans la réalité ces deux régions devraient être séparées. De plus, il ne prend pas en compte une solution optimale globale étant donné qu'il considère uniquement les optimum locaux qu'il garde à chaque étape.

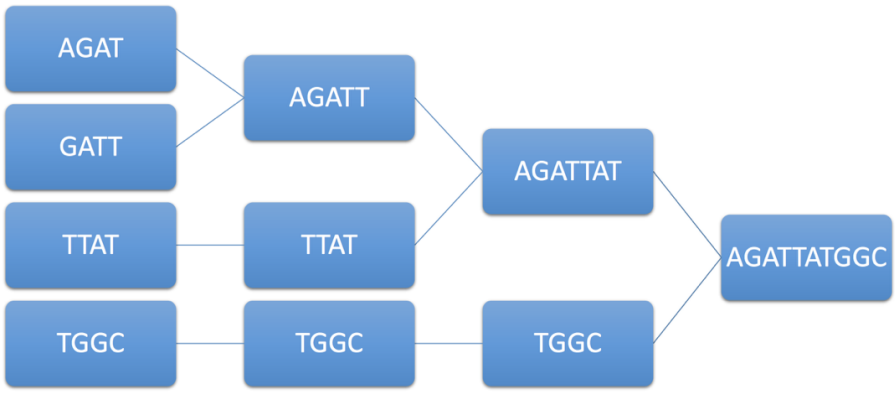

Figure 2.1 – ***Principe de fonctionnement de la méthode gloutonne.*** *Soit les reads AGAT, GATT, TTAT et TGGC. La première étape consiste à trouver la paire de read ayant le score de chevauchement le plus élevé : ici ce sont les reads AGAT et GATT dont le score de chevauchement est de 3. On assemble donc ces deux reads et on recalcule les scores de chevauchement pour voir quelle nouvelle paire sera assemblée. On continu cela jusqu'à obtenir une seule séquence. [Source : http ://data-science-sequencing.github.io/Win2018/lectures/lecture6/]*

2.2 Méthode *Overlap Layout Consensus* (OLC)

Cette méthode se base sur la construction d'un graphe de chevauchement. Elle se divise en trois étapes : *Overlap*, *Layout* et *Consensus*. (Figure 2.2)

La première étape consiste à comparer les *reads* entre eux afin de trouver des paires de *reads* qui se chevauchent (Figure 2.2, (B)). Le graphe de chevauchement est construit au fur et à mesure (Figure 2.2, (C)) : les noeuds représentent les *reads* et deux noeuds seront connectés ensembles par une arrête si il y a chevauchement entre ces *reads*.

La seconde étape consiste à rechercher dans le graphe le chemin passant par tous les noeuds permettant de reconstruire une séquence contiguë (chemin Hamiltonien, Figure 2.2, (D)). Cette étape permet de déterminer l'ordre dans lequel les reads seront assemblés en contigs.

Enfin, la dernière étape consiste, après correction des erreurs, à trouver une séquence consensus à partir des contigs obtenus à l'étape précédente (Figure 2.2, (E)).

Cette méthode est moins gourmande en mémoire que la méthode gloutonne car le graphe permet de stocker uniquement les informations essentielles sur les chevauchements. Mais la recherche d'un chemin Hamiltonien étant un problème NP-complet, des heuristiques sont alors utilisées puisqu'il n'existe à ce jour aucun algorithme capable de résoudre ce type de problème.

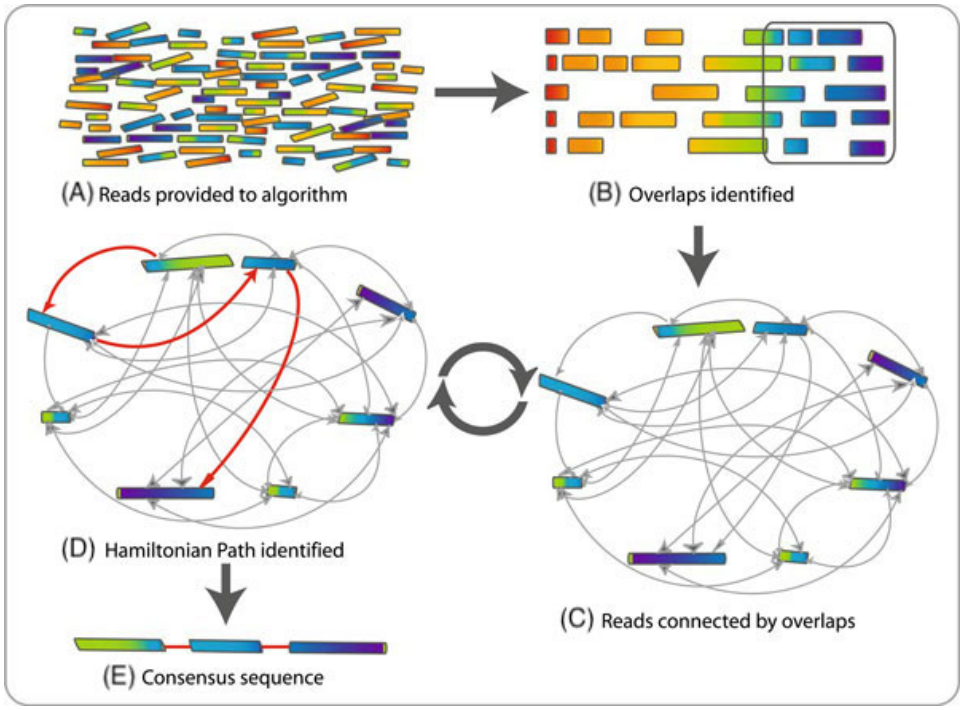

Figure 2.2 – ***Principe de fonctionnement de la méthode OLC.*** *(A) Reads donnés en entrée à l'algorithme, (B) Identification des chevauchements entre chaque reads, (C) construction d'un graphe de chevauchement, (D) recherche d'un chemin Hamiltonien, (E) construction d'une séquence consensus. [Source : J.Commins et al. Computational Biology Methods and Their Application to the Comparative Genomics of Endocellular Symbiotic Bacteria of Insects Bio.Proc.Online 2009]*

2.3 Méthode du chemin Eulérien

La méthode du chemin Eulérien utilise un graphe de De Bruijn. Ce graphe orienté représente toutes les sous-séquences chevauchantes de taille k des reads séquencés, aussi appelées k-mer.

Pour commencer, les reads sont fragmentés en k-mers. Ces k-mers seront ensuite utilisés pour construire un graphe de De Bruijn. Les noeuds constituent les préfixes et suffixes de taille k-1 de chaque k-mers et les arrêtes représentent les k-mers. Deux noeuds seront connectés uniquement s'ils partagent un chevauchement de taille k-2. Une fois le graphe construit, on chercher à trouver un chemin eulérien, c'est à dire un chemin qui passe par toutes les arrêtes du graphe. (Figure 2.3)

L'avantage de cette méthode est qu'elle permet de compresser l'information en créant le graphe de De Bruijn sachant qu'il existe des algorithmes polynomiaux permettant de trouver les chemins eulériens. Il faut tout de même noter qu'elle est très sensible aux erreurs de séquençage. En effet, les erreurs vont mener à la création de nouveaux k-mers augmentant la complexité du graphe. De plus, la décomposition des reads en k-mers mène à une perte d'information.

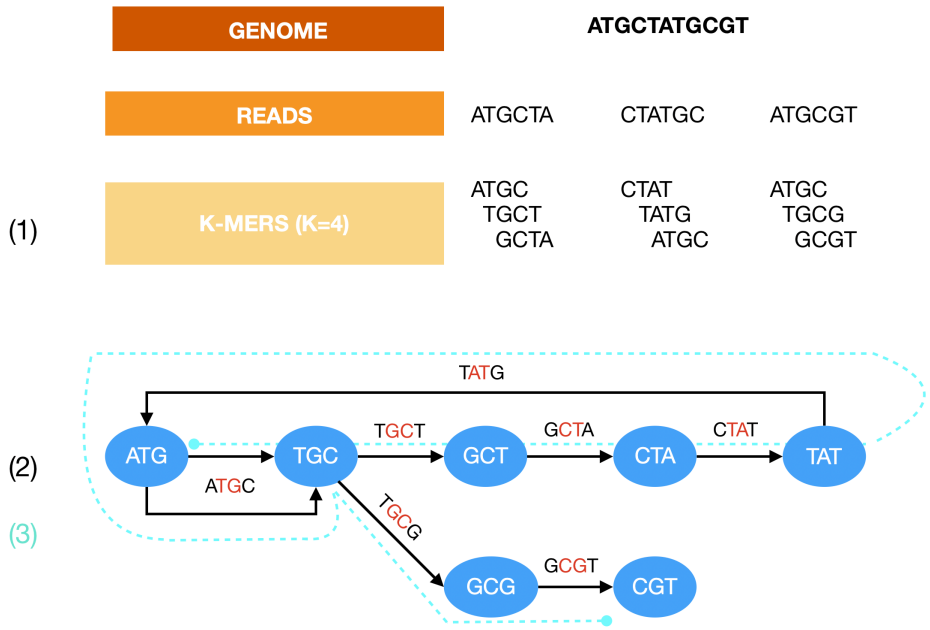

Figure 2.3 – ***Principe de fonctionnement de la méthode de chemin eulérien.*** *(1) Décomposition des reads en k-mers, (2) Construction du graphe de De Bruijn et (3) Recherche du chemin eulérien.*

2.4 Autres méthodes

Les trois méthodes présentées précédemment sont celles qui sont les plus connues et utilisées par les assembleurs. Toutefois, il existe d'autres méthodes, moins connues, utilisées par certains outils.

2.4.1 *String graph*

Le *string graph* [2, 3] est une méthode similaire à la méthode OLC. En effet, pour construire ce graphe orienté, il faut tout d'abord calculer les chevauchements entre chaque *reads*. A la différence de la méthode OLC qui utilise les *reads* dans leur totalité, cette méthode indexe préalablement les *reads* à l'aide d'une structure telle que le FM-index [1] par exemple. Le *string graph* sera construit à partir des données indexées : les noeuds, non étiquetés, représentent le début de chaque *read*. Deux noeuds seront connectées s'il y a chevauchement entre les *reads*. Les arrêtes sont étiquetées avec la séquence en amont du chevauchement du noeud de départ.

2.4.2 HGAP (*Hierarchical Genome Assembly Process*)

Cette méthode [4], uniquement implémentée pour l'assemblage de *reads* générés par les séquençages de 3ème génération, se décompose en 3 étapes :

— Pré-assemblage : les *reads* les plus longs sont choisi comme base pour aligner les autres *reads*. Pour chaque alignement, une séquence consensus sera établie.

— Assemblage : les séquences pré-assemblées sont assemblées avec un assembleur utilisant un algorithme de type OLC.

— Consensus : cette étape vise à réduire les potentielles erreurs d'assemblage pour fournir une séquence consensus constituant l'assemblage final.

1. Structure d'indexation de données basée sur la transformée de Burrows-Wheeler. Permet de compresser l'information et d'y faire des recherches plus rapidement.

3. Outils d'assemblage

Au vue du nombre élevé d'assembleurs, on trouve de nombreux recensements dans la littérature. Cette partie présentera dans un premier temps les différents outils que l'on peut rencontrer et leurs caractéristiques. Dans un second temps, les résultats d'un sondage sur l'utilisation des outils d'assemblage sera présenté. Pour terminer, une comparaison des outils basé sur différentes publications visera à déterminer les performances relatives des outils comparés.

3.1 Recensement des outils

Un total de 33 outils différents à été identifié (bien qu'il soit possible que certains outils, possiblement moins connus, aient pu passer au travers de la recherche). Comme le montre la table 3.1, les outils peuvent être distingués selon différents critères : l'année de sortie de l'outil, le type de *read* prit en charge et l'algorithme implémenté.

Si on s'intéresse aux années de sortie des outils, on remarque que sur 20 ans, un nombre constant d'outils apparaît d'années en années. Seule la nature des reads prit en compte par les outils change. En effet, jusqu'au milieu des années 2000, les outils qui ont été créés gèrent des données issues du séquençage de 2ème génération. Ensuite, les outils créés jusqu'à aujourd'hui sont capables (et spécialisés) dans la gestion de données issues du séquençage de 3ème génération. Cette coupure s'explique par les avancées du séquençage ADN.

Ensuite, en s'intéressant au type de *reads* prit en charge par les outils, on peut les séparer en quatre catégories : ceux gérant les données de type Sanger (séquençage de 1ère génération), ceux gérant les données de type *short read* (séquençage de 2ème génération), ceux gérant les données de type *long read* (séquençage de 3ème génération) et ceux capable de faire de l'assemblage hybride *short read/long read*.

Enfin, il est possible de différencier les outils suivant l'algorithme qu'il implémente. On remarque qu'une grande partie des outils implémente la méthode du graphe de De Bruijn.

Table 3.1 – *Caractéristiques des outils d'assemblage de génome*

Outil	Sortie	Type de *reads*	Algorithme
Flye	2019	L	Repeat graph
Peregrine	2019	L	Sparse HIereachical MimiMizER (SHIMMER)
Ra	2019	L	OLC
Shasta	2019	L	?
Wtdbg2	2019	L	FBG (Fuzzy Bruijn Graph)
Canu	2017	L	OLC
Smartdenovo	2017	L	OLC
Unicycler	2017	S,L,H	GDB/OLC
Miniasm	2016	L	OLC
NOVOPlast	2016	S	Extension par graines
IVA	2015	S	OLC

Suite à la page suivante

TABLE 3.1 – *Caractéristiques des outils d'assemblage de génome*

Outil	**Sortie**	**Type de *reads***	**Algorithme**
Megahit	2015	S	GDB
Falcon	2013	L	HGAP
Masurca	2013	S,H	OLC/GDB
SGA	2012	S	String graph
Spades	2012	S	GDB
IDBA	2012	S	GDB
Mapsembler	2012	S	OLC
Metavelvet	2012	S	GDB
Minia	2012	S	GDB
Newbler	2012	Pyro	OLC
IMR/DENOM	2011	S	?
Locas	2011	S	OLC
Soapdenovo	2009	S	GDB
Abyss	2009	S	GDB
Allpaths	2008	S	GDB
Euler-SR	2008	S	GDB
Velvet	2008	S	GDB
Edena	2008	S	OLC
SSAKE	2007	S	Glouton
VCAKE	2007	S	Glouton
Euler	2001	Sanger	GDB
CELERA	2000	Sanger	OLC

Sigles : S : *short read*, L : *long read*, H : hybride, GDB : graphe de De Bruijn, OLC : *Overlap Layout Consensus*.

La figure 3.1 résume les outils présents dans la table 3.1. Elle sépare les outils suivant le type de *reads* géré par l'assembleur et l'algorithme qu'il implémente. Cette figure a pour objectif d'aider à prendre une décision quant au choix de l'assembleur à utiliser pour réaliser un assemblage suivant les données à disposition.

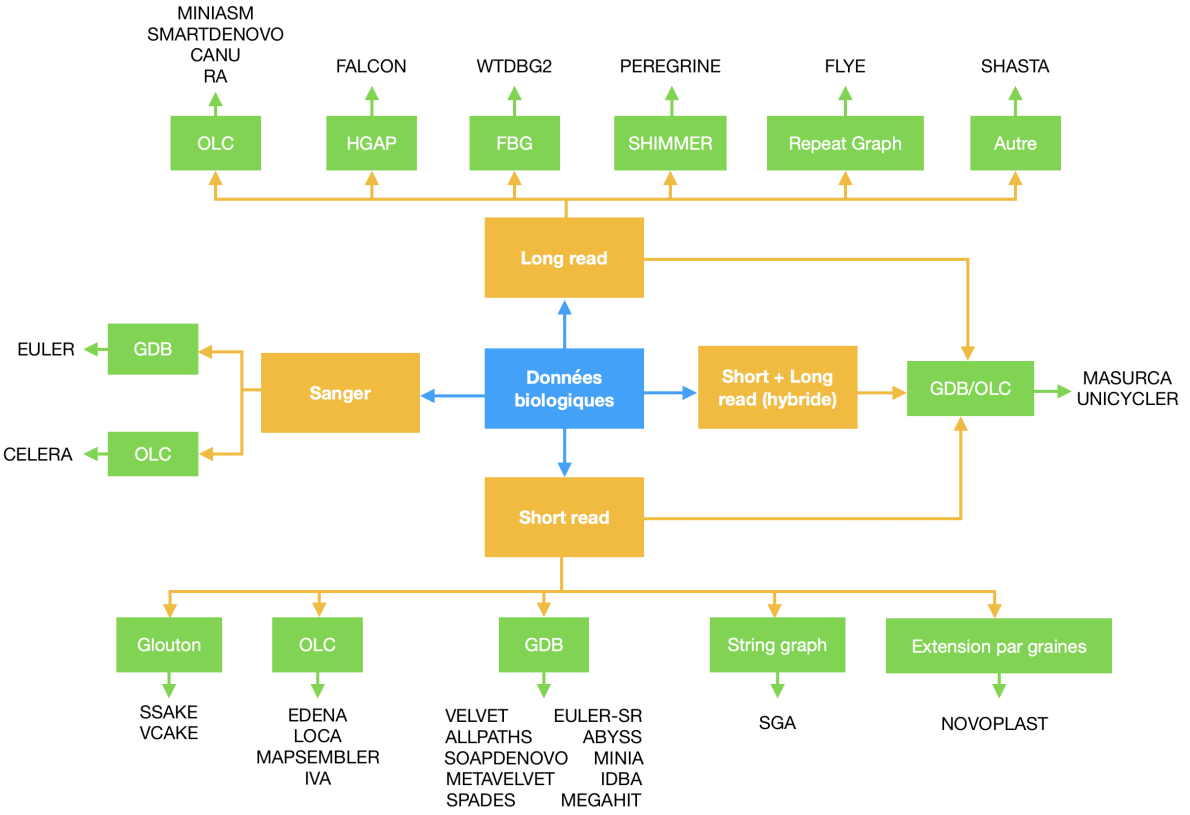

FIGURE 3.1 – ***Guide d'aide à la décision dans le choix d'un assembleur en fonction des données biologiques à disposition et du type d'algorithme implémenté.***

3.2 Sondage

Afin d'élargir la recherche bibliographique, un petit sondage de 8 questions destiné aux bioinformaticiens ayant déjà assemblé un génome à été diffusé dans différentes structures de recherche à Montpellier. L'objectif de ce sondage était d'avoir un aperçu des outils d'assemblage les plus utilisés dans le quotidien des professionnels, dans quel contexte étaient-ils utilisés et quels étaient leur points forts ou faibles selon eux. Les questions posées se trouvent dans la figure 5.2 en annexe. Ce sondage a comptabilisé 15 réponses.

Concernant la première question du sondage, le premier objectif était d'établir quels assembleurs étaient connus ou non parmi ceux proposés afin de voir si certains étaient plus connus que d'autres. Le second objectif était de voir si d'autres assembleurs étaient connus des bioinformaticiens.

Sur 16 outils proposés, seuls 2 n'étaient pas connus des professionnels ayant répondu au sondage. De plus, un total de 16 outils est venu compléter la liste des assembleurs proposés dans cette question.
En s'intéressant aux 32 assembleurs, 6 d'entre eux (Canu, Spades, SOAPdenovo, AllPaths, Abyss et Velvet) sont ressortis du lot puisque plus de la moitié des personnes les connaissaient.

Ensuite, si on s'intéresse aux outils les plus utilisé, aucun ne se démarque plus qu'un autre. Chaque personne en utilise un en particulier voir plusieurs pour certains. Cela s'explique par

les réponses données dans la suite du sondage. En effet, leur choix est dû notamment au type des données utilisées : certains utilisent des données *short read* issues d'un séquençage de 2ème génération et d'autres des données *long read* issues d'un séquençage de 3ème génération. De plus, tous ne travaillent pas sur les mêmes types de génome (bactériens, mammifères, plantes ...).

Pour terminer, ce qui ressort de ce sondage est que chaque outil à ses points forts et ses points faibles, et que le choix d'en utiliser un plutôt qu'un autre s'effectue en fonction des données dont disposent les bioinformaticiens.

3.3 Comparaison des outils

L'objectif de cette section est d'étudier des comparaisons de différents assembleurs ayant déjà été réalisées afin de déterminer si un outil est globalement "meilleur" ou non. Pour cela, l'intérêt s'est porté sur deux publications différentes : une comparant cinq assembleurs *long read* et une autre comparant sept assembleurs *short read*.

Comparaison des assembleurs *long read* :

Début 2019, Ryan R. Wick et Kathryn E. Holt [5] se sont intéressés à la comparaison de cinq assembleurs *long read* : Canu, Flye, Ra, Unicycler et Wtdbg2. Pour cela, ils ont utilisés des données réelles et simulées d'un génome bactérien. Chaque outil à été testé avec les paramètres par défaut ou recommandés dans la documentation.

Les outils ont été comparés selon quatre critères :

- La robustesse de l'assemblage : pour cela, un set de reads de mauvaise qualité à été simulé et l'influence de 7 paramètres sur la qualité de l'assemblage à été testé.
- La fiabilité : pour cela, la capacité à reconstruire un chromosome en entier à été testé à partir d'un bon jeu de données.
- L'identité de séquence.
- Le temps mit pour réaliser l'assemblage.

Les résultats de cette comparaison ont montré qu'aucun outils n'est meilleur que l'autre, mais que chacun à ses qualités et ses défauts. Globalement, chacun des outils est relativement fiable et robuste, bien que Flye et Ra le sont légèrement plus que les autres outils testés. Les différences entre les outils se portent essentiellement sur le temps mit pour faire l'assemblage et la capacité à bien circulariser le génome. Concernant le temps d'exécution, Canu est le plus lent : l'assemblage durent plusieurs heures contrairement aux autres outils qui assemblent en moins d'une heure. Pour la circularisation du génome, Ra est moins bon par rapport aux autres outils car il supprime une centaine de base, et Flye supprime également quelques base. Unicycler est l'outil qui circularise parfaitement le génome.

Il faut tout de même noter que ces résultats portent sur l'assemblage de génome bactérien. Il est donc possible que les outils aient des performances différentes dans le cas d'assemblage de génomes eucaryotes qui sont linéaires et non circulaires.

Comparaison des assembleurs *short read* :

En 2011, Lin et al. [6] ont réalisé une étude visant à comparer plusieurs outils d'assemblage *short read* de génome *de novo*. Ils se sont intéressés à sept outils différents pour lesquels les paramètres par défaut ou recommandés dans la documentation ont été utilisé : SSAKE, VCAKE, Euler-sr, Edena, Velvet, ABySS et SOAPdenovo.

Pour tester les outils, différentes données ont été utilisées :

— Données réelles : 8 séquences récupérées sur NCBI, dont la taille varie entre 99 kb à 100 Mb et le contenu en GC est différent pour chaque séquence.

— Données simulées : des *reads paired-end* et *single-end*, pour lesquels la profondeur, le BCER [1] et la longueur des reads varient, ont été simulés à partir des séquences récupérées sur NCBI.

Afin de comparer les performances des outils, les critères suivant ont été utilisés :

— Le N50 : taille du contig à partir duquel au moins 50% du génome est couvert par l'assemblage.

— Le pourcentage de séquence couvert par les contigs assemblés.

— La justesse de l'assemblage : elle est basée sur le taux d'erreur, qui correspond au rapport du nombre de *mismatch* (déterminé par l'alignement des contigs aux séquences initiales) sur le nombre total de bases des contigs alignés.

— La mémoire requise et le temps nécessaire pour faire l'assemblage.

Les résultats de cette étude montrent que les performances des outils testés varient suivant les propriétés des données utilisées. Aucun outil ne surpasse un autre. Ainsi, le choix de l'outil dépendra donc des propriétés des *reads* à assembler. Par exemple, SSAKE, Edena et Euler-sr ont besoin d'une profondeur plus élevée que les autres outils afin d'obtenir un N50 élevé (une des caractéristiques définissant la bonne qualité d'un assemblage). L'utilisateur doit donc prendre en compte chaque spécificité des outils afin de choisir celui qui sera le plus adapté aux données et ressources informatiques dont il dispose.

1. *base calling error rate*

4. Conclusion

Ce projet bibliographique avait pour but de donner une vue d'ensemble des nombreux outils d'assemblage *de novo* existant et potentiellement de donner un guide d'aide à la décision, c'est à dire quel outil utiliser suivant le type de donnée à disposition. On distingue parmi les différents outils deux grands types : il y a ceux gérant les données de séquençage de seconde génération, appelées *short read* et ceux gérant les données de séquençage de 3ème génération, appelées *long reads*.

Malheureusement, le manque de temps pour réaliser cette étude bibliographique à fait que cette étude manque de profondeur et n'est pas aboutie, terminée. Il aurait fallu parcourir plus en détail chaque outil pour déterminer leurs spécificités, étudier plus de publications comparant les outils pour déterminer leurs points faibles et leurs points fort et approfondir la séparation des outils suivant l'utilisation que l'on souhaite en faire afin de réaliser un guide d'aide à la décision complet.

En conclusion, ce rapport est une ébauche, un début d'étude bibliographique qu'il serait intéressant de poursuivre et d'approfondir.

5. Annexes

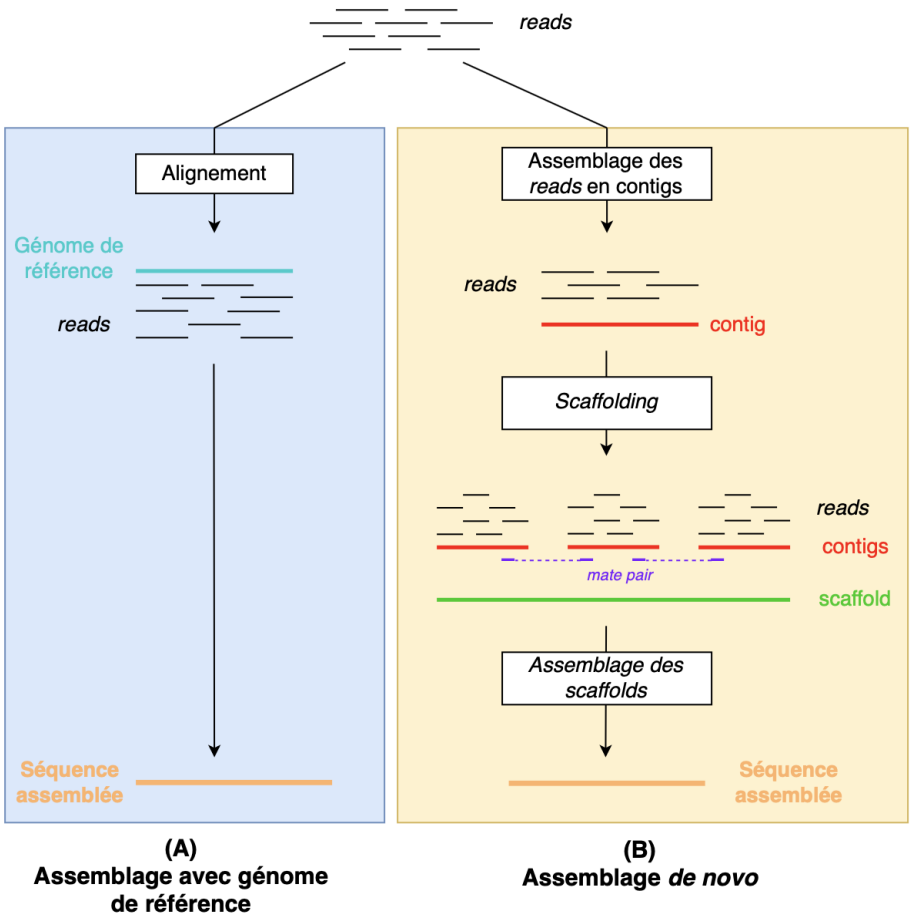

FIGURE 5.1 – *Les différents types d'assemblage.*

1. Quel(s) outil(s) d'assemblage connaissez-vous ? (Si vous en connaissez d'autres qui ne sont pas dans la liste ci-dessous, les ajouter dans la case "autre").
Propositions : Canu, SPAdes, Minia, SOAPdenovo, AllPaths, ABySS, Velvet, Euler-sr, Shorty, Edena, CABOG, Celera, Newbler, VCAKE, SHARGCGS, SSAKE, Autre(s).

2. Parmi ces outils, lequel utilisez-vous le plus régulièrement ?

3. Pourquoi utilisez-vous cet outil plutôt qu'un autre ?

4. Sur quel(s) type(s) de données et sur quel(s) génome(s) l'utilisez-vous ?

5. Quels sont les points forts de cet outil ?

6. Quels sont les points faibles de cet outil ?

7. Quels paramètres utilisez-vous ? (Paramètres par défaut ou paramètres plus spécifiques).

8. Si vous avez des commentaires ou références à ajouter, veuillez les écrire ci-dessous. Si vous souhaitez être informé des résultats de cette enquête, vous pouvez laisser en plus votre adresse mail.

Figure 5.2 – ***Questions posées dans le sondage sur l'utilisation des outils d'assemblage de génome (8 questions).***

Bibliographie

[1] Sutton G. Miller JR, Koren S. Assembly algorithms for next-generation sequencing data. *Genomics*, 95 :315–327, 2010.

[2] Durbin R. Simpson JT. Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.*, 22 :549–556, 2012.

[3] Joseph Henson, German Tischler, and Zemin Ning. Next-generation sequencing and large genome assemblies. *Pharmacogenomics*, 13 :901–15, 2012.

[4] Alexander D. Marks P. et al. Chin, C. Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nat Methods*, 10 :563–569, 2013.

[5] Kathryn E. Holt Ryan R. Wick. Benchmarking of long-read assembly tools for bacterial whole genomes. `https://github.com/rrwick/Long-read-assembler-comparison/tree/96dfbe7e6edd6195cdd7fbe6f532f0022a7ebbb9`. Dernier accès : 26-12-2019.

[6] Shen H Zhang L Papasian CJ Deng HW Lin Y, Li J. Comparative studies of de novo assembly tools for next-generation sequencing technologies. *Bioinformatics*, 27 :2031–2037, 2011.