HLIN608 : Algorithmique du Texte
Assemblage

1 Introduction

Dans cette partie, nous allons parler de l'*assemblage*.

1.1 Génomique

Le *séquençage* est un processus qui, à partir d'une molécule d'ADN, permet de déterminer l'information génétique, autrement dit l'ordre des paires de bases (nucléotides) qui constituent la molécule. Les applications sont multiples, notamment en santé, en biologie, paléontologie, police scientifique...

C'est l'avènement des séquenceurs automatiques dans les années 80 qui a permis la naissance de la génomique, et notamment de la génomique informatique. La taille du génome à séquencer est très variable : il peut s'agir de quelques millions de paires chez les bactéries, à plusieurs milliards chez la plupart des animaux et des plantes.

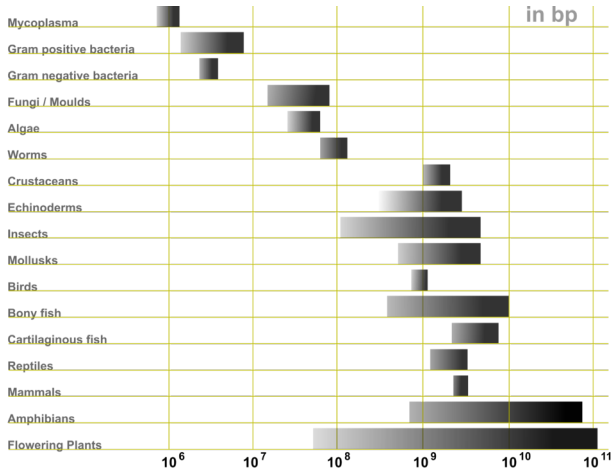


Figure 1: La taille du génome est très variable selon les organismes.

Le séquençage doit se faire à l'aide de séquenceurs automatiques, mais le problème de ceux-ci est qu'ils ne peuvent proposer que des séquences génomiques ne contenant que quelques centaines de bases à la fois. On se retrouve ainsi avec un génome parcellaire et un traitement informatique est nécessaire pour reconstituer la séquence complète. On peut voir cela comme un immense puzzle à assembler.

1.2 Historique

- **1973 :** première publication d'une séquence de 24bp;

5'– T G G A A T T G T G A G C G G A T A A C A A T T **–3'**
| | | | | | | | | | | | | | | | | | | | | | | | |
3'– A C C T T A A C A C T C G C C T A T T G T T A A **–5'**

- **1977-80 :** méthode de séquençage Sanger, qui permit à Wally Gilbert et Fred Sanger d'obtenir le prix Nobel ;
- **1983 :** développement de la technique *PCR* (pour "polymerase chain reaction") : cette technique permet de créer de multiples copies de fragments d'ADN, ce qui a permis par la suite de créer des séquenceurs automatiques ;
- **1987 :** premier séquenceur automatique : *Applied Biosystems Prism 373*;
- **1998 :** génome de l'organisme *C. elegans séquencé*;
- **2000 :** génome humain séquencé.

Dans le milieu des années 2000, une nouvelle génération de séquenceurs arrive sur le marché (NGS, pour "next generation sequencing"). Ces séquenceurs permettent de générer des millions de séquences avec un coût faible. Notamment, on peut voir :

- **2005 :** la société 454 Life Sciences (rachetée par Roche) produit le premier séquenceur NGS ;
- **2006 :** séquenceur *Illumina* de chez Solexa ;
- **2007 :** séquenceur *SOLiD* de chez Applied Biosystems ;
- **2011 :** séquenceur *PGM* de chez Ion Torrent ;

On peut également distinguer une autre génération apparaît dans les années 2010 permet de créer des séquences encore plus longues.

- **2009 :** premier séquenceur "monomolécule" de chez Helicos Genetic Analyser System ;
- **2011 :** premier séquenceur de longue lectures de chez Pacific Biosciences, appelé PacBio RS System ;
- **2012 :** Oxford Nanopore Technologies sort son séquenceur à ultralongue molécule.

2 Séquençage

Le principe du séquençage est le suivant : tout d'abord, on fait de multiples le matériel génétique à l'aide de la technique PCR (qui est en fait une réaction chimique). On appelle cela l' *amplification* de l'ADN. On casse ensuite cette molécule amplifiée en millions de fragments aléatoires. Dans ces fragments, appelés *lectures* (reads en anglais), l'ordre des bases est connu.

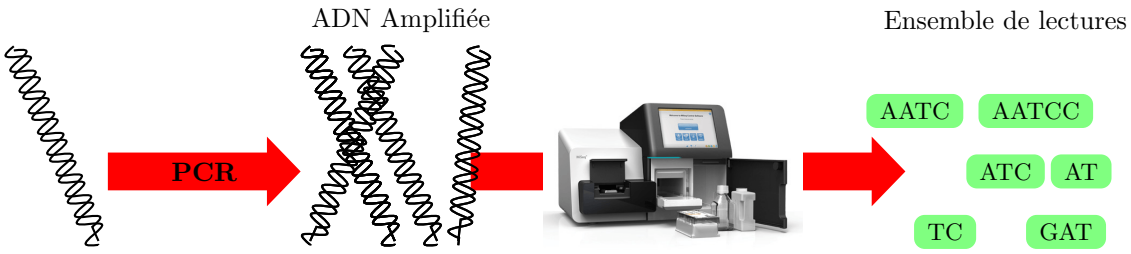


Figure 2: Étapes du séquençage

Seulement, on obtient alors que des séquences partielles. Si l'on veut obtenir la séquence originale, il faut "assembler" les lectures obtenues. Il n'est pas possible de la faire à la main car le nombre de lectures est trop important. Heureusement, on peut utiliser les chevauchements entre les lectures pour pouvoir reconstituer la séquence complète : en effet, comme on a amplifié la molécule d'ADN, chaque sous-séquence apparaît plusieurs fois dans l'ensemble de lectures et comme le matériel issu de cette amplification est cassé de façon aléatoire, cela veut dire que certaines lectures vont partager des sous-séquences communes. Ainsi, si par exemple on considère deux lectures $L_1 = AATCGA$ et $L_2 = TCGAGA$, les quatre dernières bases de L_1 correspondent aux quatre premières bases de L_2. On peut donc suspecter que la sous-séquence $AATCGAGA$ apparaît dans la séquence originale. Un exemple est donné par la Figure 3 : dans cet exemple, la séquence en rouge correspond à la séquence originale (elle est bien entendu inconnue) et les sous-séquences en vert correspondent aux lectures issues du séquençage.

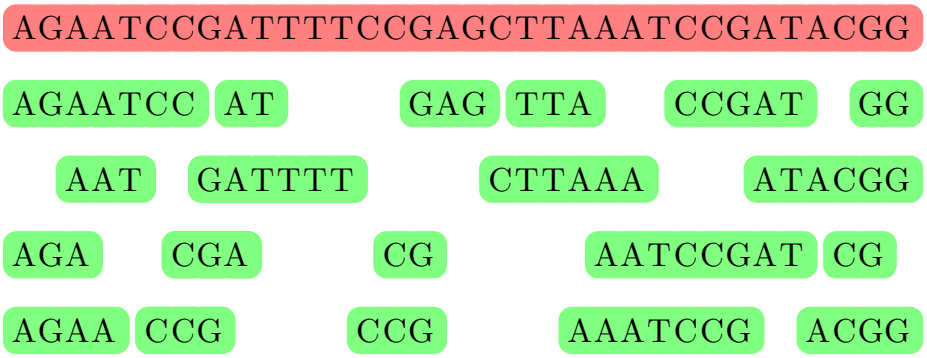


Figure 3: Assemblage

On définira formellement le problème de l'assemblage un peu plus loin dans ce document. Notons auparavant quelques limitations de cette méthode.

- La première est que les séquenceurs automatiques ne sont "exacts", c'est-à-dire qu'ils peuvent faire des erreurs de lecture : ainsi au sein d'une lecture, un nucléotide peut être remplacé par un autre (ATC à la place de AGC), un nucléotide peut ne pas avoir été lu ($AATC$ à la place de $AAGTC$) ou un nucléotide peut avoir été ajouté alors qu'il n'existe pas ($AGTTC$ à la place de $ATTC$). Ces erreurs de lecture peuvent être en partie compensées par le PCR : une erreur a peu de chance d'apparaître dans toutes les copies du brin d'ADN d'origine.
- Certaines zones du génome ne peuvent pas être séquencée correctement : c'est le cas notamment des parties de chromosomes "repliées" comme les centromères ou télomère.
- Enfin, certaines parties de la séquences originelle se répètent dans celle-ci. Cela fausse la recherche de chevauchements. Pour se rendre compte de cela, on peut regarder la Figure 4 : en haut, la séquence contient deux répétitions (A1 et A2) d'une même sous-séquence. En pratique, avec les outils d'assemblage classiques utilisant les chevauchements, toutes les lectures appartenant à ces deux répétitions seront regroupées au même endroit (comme en bas de la figure), et ces séquences répétées auront du coup du mal à s'intégrer au sein du reste de la séquence. En pratique, dans la plupart des modèles, on a tendance à ignorer ces séquences répétées (ou au moins à ne considérer qu'elles n'apparaissent qu'une seule fois).

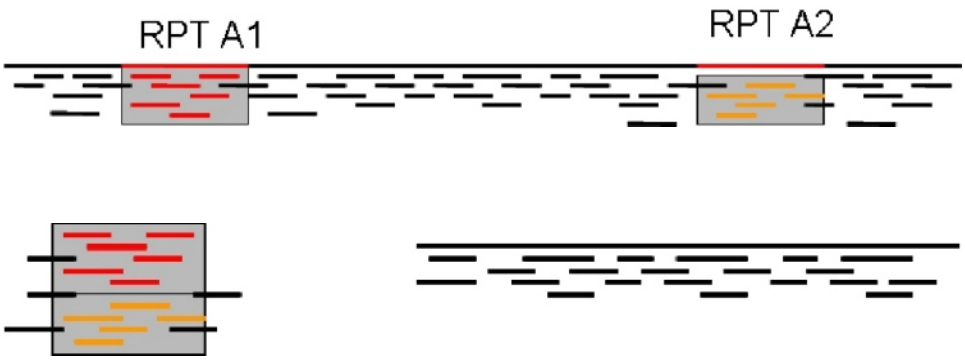
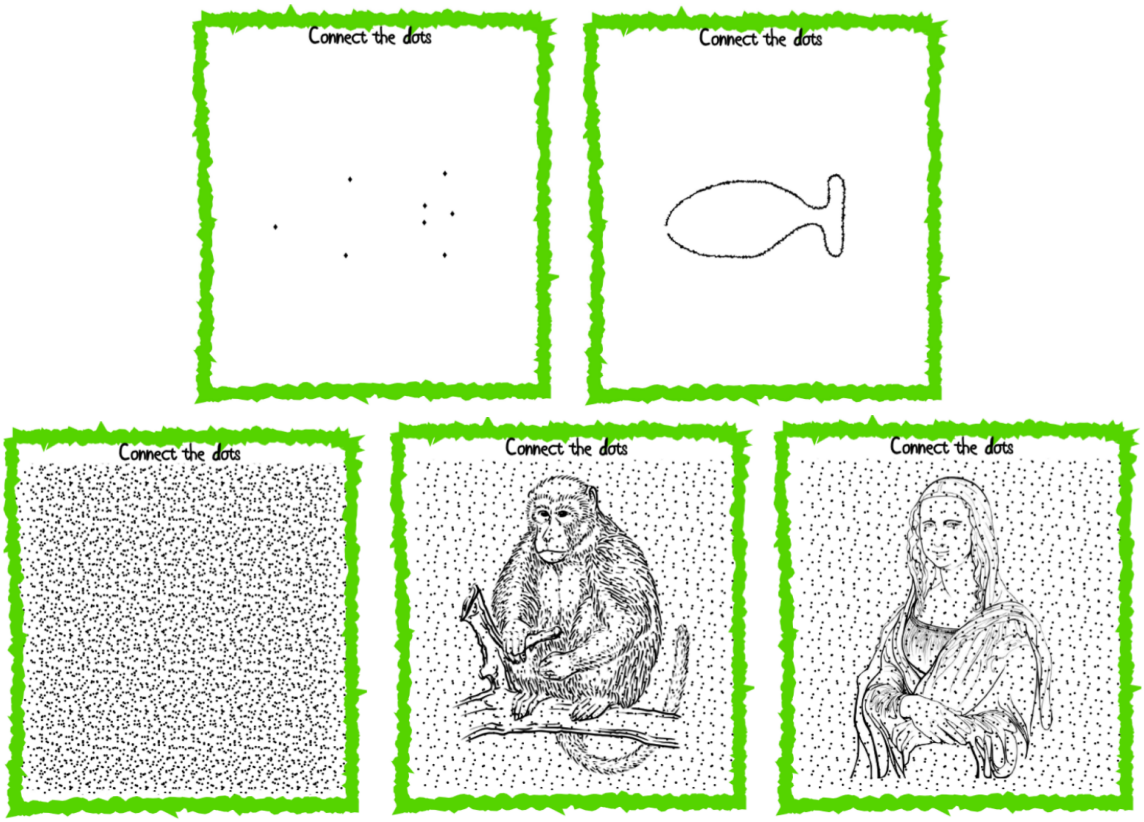


Figure 4: Répétitions

3 Assemblage

En entrée de l'*assemblage*, on a donc un puzzle gigantesque composé de millions de grosses pièces (short reads en anglais) contenant peu d'erreurs, ou de grosses pièces (long reads) contenant davantage d'erreur. On peut parfois, on a un modèle approximatif pour aider à assembler les lectures avec un génome de référence. Parfois, ce modèle n'existe pas, on parle alors d'assemblage *de novo*. Gardons à l'esprit que la difficulté vient surtout du fait qu'il y

ait énormément de lectures à assembler. On peut tenter de faire un parallèle avec les dessins suivants : en haut, la solution est facile à trouver car il y a peu de points à relier, mais en bas, il y a plusieurs façon de relier les nombreux points tout en sortant quelque chose de cohérent.



3.0.1 Un brin de théorie

On note P, la *profondeur* de couverture du génome. On a :

$$P = \frac{\sum \texttt{longueurs des fragments.}}{\texttt{Taille du génome}}$$

On peut voir la profondeur comme le nombre de fois qu'on peut recouvrir la séquence originelle avec toutes les lectures, sans faire correspondre les bases. Par exemple, si on a un génome de 5Mb et qu'on obtient des données séquencées de taille 25Mb, alors la profondeur est de 5. Et on la note $5X$. Pour un autre exemple, plus visuel, on peut regarder la Figure 3 : on voit que l'assemblage ici permet de faire quatre "étages", comme ces étages ne sont pas complétement remplis (il y a des trous), on a une profondeur située entre 3 et 4. Plus la profondeur est grande, plus le génome est couvert : il y a moins de chance qu'une partie du génome ne soit présente sur aucun étage. Pour avoir une profondeur plus grande, il faut générer davantage de données séquencées. Cependant, cela coûte plus cher. D'un point de vue économique, il peut être parfois plus intéressant de combler les trous de façon un peu plus ciblée. Il existe une profondeur optimale qui est un équilibre entre le coût et la qualité du génome. Pour déterminer une profondeur optimale, on note :

- n la longueur moyenne des lectures ;
- L la longueur de la séquence originelle ;
- K le nombre de lectures .

On suppose que n est très petit devant L et que $\frac{P \times L}{n}$ est très grand. La théorie de Lander et Waterman indique que l'on peut modéliser l'apparition d'une base dans un nombre de lectures donné par une loi de Poisson. Autrement dit, la probabilité qu'une base apparaisse dans x lecture est égale à :

$$\frac{P^x e^{-P}.}{x!}$$

On peut mettre en évidence les statistiques suivantes :

- la probabilité qu'une base n'apparaisse dans aucune lecture (*i.e.* si $x = 0$) est de e^{-P} ;
- la proportion du génome couverte est égale à $1 - e^{-P}$;
- longueur totale des trous est de Le^{-P} (la somme des probabilités pour chaque base de ne pas apparaître) ;
- nombre total de trous Ke^{-P} ;
- taille moyenne des trous $\frac{n}{P}$;
- taille moyenne des parties couvertes de la séquence originelle (*contigs*) $\frac{n}{P} \times e^{P}$.

Pour se rendre compte de l'importance de la profondeur, regardons un exemple concret : on considère le génome humain, avec $L = 3.10^9$ et $n = 1000$.

- si on génère $K = 9 \cdot 10^9$ lectures, on a une profondeur $3X$. On a $1 - e^{-3} = 95\%$ du génome couvert, il y a $9 \cdot 10^6 \times 0.05 = 450000$ trous de longueur moyenne de $1000/3 = 333$ bases;
- si on génère $K = 3 \cdot 10^{10}$ lectures, on a une profondeur $10X$. On a $1 - e^{-10} = 99.99\%$ du génome couvert, il y a $3 \cdot 10^7 \times 5 \cdot 10^{-5} = 1500$ trous de longueur moyenne de $1000/10 = 10$ bases ;

3.1 Plus courte séquence commune

Pour résoudre le problème de l'assemblage à l'aide d'algorithmes, on le formalise comme un problème combinatoire sur des mots. Ce problème est appelé Plus courte séquence commune (Shortest Superstring Problem).

Shortest Superstring Problem (SSP)	
Entrée :	ensemble de mots $\mathcal{F} = \{F_1, \ldots, F_n\}$
Sortie :	un séquence S de longueur minimum telle que chaque mot de $\mathcal{F}$ est un sous-mot de S.

Exemple

```
F1 :  AATGCC
F2 :     GCCTTACAC
F3 :          ACACTG
F4 :            ACTGAAGG
F5 :               GAAGGTTTA
---------------------------
B  :  AATGCCTTACACTGAAGGTTTA
```

Le problème est NP-difficile dès que l'ensemble de mots contient davantage que trois mots sur un alphabet de plus de deux lettres (la preuve de la NP-difficulté est faite en TD). Cela signifie qu'il n'existe a priori pas d'algorithme polynomial pour résoudre SSP (sauf si P=NP). Rappelons qu'on peut résoudre un tel problème en utilisant un algorithme au moins en temps exponentiel, mais dans ce cas, le temps nécessaire pour résoudre les grosses instances devient trop important en temps humains (plusieurs milliers voir plusieurs millions d'années). Pour résoudre SSP en temps polynomial, on va devoir utiliser des *heuristiques*. C'est à dire des algorithmes qui ne retournent pas toujours une solution optimale, mais retourne une solution qu'on espère en principe proche d'une solution optimale.

4 Méthode Gloutonne

Une heuristique très naturelle est la méthode *gloutonne*. Un algorithme glouton est un algorithme qui va faire effectuer un choix à chaque étape optimisant un critère particulier, et ne revient jamais sur ces choix. Concrètement, si on l'applique une méthode gloutonne à SSP, on peut chercher à optimiser les scores de chevauchement (appelé alignement semi-global). Le score de chevauchement entre deux mots est égal à la longueur du plus long préfixe d'un mot correspondant à un suffixe de l'autre. Par exemple considérons deux mots $F_1 = ACCTGA$ et $F_2 = TGATTGC$, le score d'alignement noté, $overlap(F_1, F_2)$, est égal à 3 car les trois dernières lettres de F_1 sont les trois premières lettres de F_2. Notons que $overlap(F_1, F_2) \neq overlap(F_2, F_1)$. La méthode gloutonne sur SSP peut être décrite par ses deux étapes, comme suit.

Méthode Gloutonne.

(a) calculer les scores de chevauchement entre chaque paire de mot ;

(b) sélectionner une paire avec le meilleur score de chevauchement et fusionner ces deux mots. Recommencer.

Exemple : Déroulons cet algorithme sur un petit exemple contenant les cinq mots $F_1 = ACCTGA, F_2 = TGATTGC$, $F_3 = GCAGC, F_4 = AGCAA$ et $F_5 = CAATG$.

Étape 1 : (a) Tableau de chevauchement :

	F_1	F_2	F_3	F_4	F_5
F_1		3	0	1	0
F_2	0		2	0	0
F_3	0	0		3	1
F_4	1	0	0		3
F_5	0	2	1	0	

(b) Un chevauchement maximum est celui entre F_1 et F_2, en les fusionnant, on obtient un nouveau mot $F_{1,2} = ACCTGATTGC$.

Étape 2 : (a) Tableau de chevauchement :

	$F_{1,2}$	F_3	F_4	F_5
$F_{1,2}$		2	0	0
F_3	0		3	1
F_4	1	0		3
F_5	0	1	0	

(b) Un chevauchement maximum est celui entre F_3 et F_4, en les fusionnant, on obtient un nouveau mot $F_{3,4} = GCAGCAA$.

Étape 3 : (a) Tableau de chevauchement :

	$F_{1,2}$	$F_{3,4}$	F_5
$F_{1,2}$		2	0
$F_{3,4}$	1		3
F_5	0	0	

(b) Un chevauchement maximum est celui entre $F_{3,4}$ et F_5, en les fusionnant, on obtient un nouveau mot $F_{3,4,5} = GCAGCAATG$.

Étape 4 : (a) Tableau de chevauchement :

	$F_{1,2}$	$F_{3,4,5}$
$F_{1,2}$		2
$F_{3,4,5}$	0	

(b) Un chevauchement maximum est celui entre $F_{1,2}$ et $F_{3,4,5}$, en les fusionnant, on obtient un nouveau mot $F_{1,2,3,4,5} = ACCTGATTGCAGCAATG$.

L'algorithme glouton renvoie la séquence $ACCTGATTGCAGCAATG$ de longueur 17. Sur cet exemple, on a trouvé une solution optimale, mais ce n'est pas toujours le cas. On peut calculer la complexité de cet algorithme. Le chevauchement entre deux mots F_1 et F_2 se calcul en $\mathcal{O}(|F_1| \times |F_2|)$ trouver le meilleur chevauchement se fait en temps $\mathcal{O}(|\mathcal{F}|^2)$. Ainsi, si on note n la taille de l'ensemble de mots et m la longueur maximum des mots de $\mathcal{F}$, on a une complexité en temps de $\mathcal{O}(n^2 \times m)$.

L'avantage de la méthode gloutonne est qu'elle est facilement implémentable et qu'elle donne des résultats assez satisfaisants. Malheureusement, elle nécessite un espace mémoire conséquent ce qui fait qu'elle n'est pas applicable sur de gros génomes : il faut en effet stocker tous les chevauchements entre les mots (il est possible de ne pas le faire, mais il faut les recalculer à chaque étape, ce qui est coûteux en temps). Le critère de chevauchement local est également généralement peu adapté pour tenir compte des répétitions.

5 Overlap-Layout-Consensus

Une autre approche pour résoudre l'assemblage est d'utiliser les outils de la théorie des graphes. On présente ici une méthode appelée *overlap-layout-consensus* qui utilise un graphe modélisant les chevauchements entre les mots. Le nom de cette méthode contient les trois étapes qui la compose :

- **Overlap :** construire un graphe orienté pondéré, où chaque sommet est un mot et chaque arc est pondéré par la valeur de chevauchement ;
- **Layout :** organiser les lectures chevauchantes en une séquence contiguë. Cela se fait trouvant un chemin orienté passant par tous les sommets (*chemin hamiltonien*) ;
- **Consensus :** corriger les éventuelles erreurs, générer des séquences consensus c'est à dire des séquences contenant un nombre important de chevauchement pour chaque lettre qui la compose.

5.1 Overlap

Soit $\mathcal{F} = \{F_1, \ldots, F_n\}$ un ensemble de lectures. On construit le graphe de chevauchement (G, ω) où G est un graphe complet dirigé et $\omega : A(G) \mapsto \mathbb{N}^+$ une fonction de poids sur les arcs de G. On a :

- l'ensemble des sommets de G, $V(G) = \mathcal{F}$ est constitué des mots de $\mathcal{F}$;
- pour chaque paire de mot (F_i, F_j), on donne $\omega(F_iF_j) = overlap(F_i, F_j)$.

Notons que le graphe obtenu est simplifiable, on peut par exemple enlever au préalable les mots qui sont entièrement contenus dans un autre mot. On peut également enlever les arcs pondéré à zéro, mais dans ce cas, la phase Layout ne contient pas forcément un chemin hamiltonien car tous les graphes n'en contiennent pas forcément.

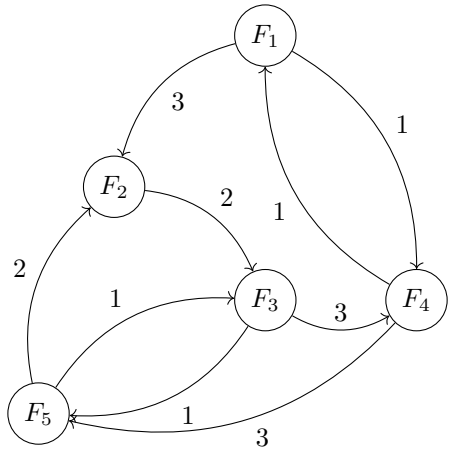


Figure 5: Graphe de chevauchement correspondant à l'exemple donné plus haut. On peut obtenir la même solution que pour l'algorithme glouton avec le chemin hamiltonien $\{F_1, F_2, F_3, F_4, F_5\}$.

5.2 Layout

Dans le graphe de chevauchement construit précédemment, on va chercher un chemin hamiltonien de poids maximum. Il s'agit d'une variante de voyageur de commerce appelée "asymétrique" ou "orientée" où il faut maximiser le coût du chemin. Ce problème est bien évidemment NP-difficile. On peut envisager plusieurs approches pour tenter de le résoudre. D'abord regardons les méthodes exactes.

- **Force brute :** tester tous les tours possibles. Dans l'arbre de recherche, l y a n possibilités pour choisir le sommet en première position, $n-1$ possibilités pour le sommet en deuxième position, $n-k$ possibilités pour le sommet en k^{ieme} position... Cela donne une complexité factorielle. Cela est encore une fois une méthode non applicable pour des génomes de grande taille.

- **Programmation dynamique :** le principe est de diviser l'instance en sous-instances plus petites et utiliser les résultats dans ces instances pour calculer une solution dans l'instance d'origine. Une stratégie peut par exemple être d'ajouter successivement les sommets dans le graphe jusqu'à obtenir l'instance complète. Le meilleur algorithme de programmation dynamique sur ce genre de problème a une complexité en $\mathcal{O}(n2^n)$, qui est bien meilleure que l'algorithme brute force, cependant sa complexité en mémoire est bien plus grande.
- **Algorithme de Séparation et évaluation (branch and bound) :** l'idée de cette méthode est de ne pas continuer à explorer l'arbre de recherche si l'on sait que l'on ne pourra de toute façon pas battre la meilleure solution explorée. Concrètement, avant de créer une nouvelle sous-branche, on va appeler une fonction qui va donner une borne sur la valeur de cette sous-branche. Cet algorithme est particulièrement efficace dans les instances du voyageur de commerce où l'inégalité triangulaire est respectée. Pour l'assemblage, trouver une fonction donnant une borne est plus difficile.
- **Programmation linéaire en nombre entier (ILP) :** il s'agit de modéliser le problème sous forme d'équations en nombres entiers puis d'envoyer ce modèle à un solveur spécialisé pour le résoudre. Ces solveurs sont souvent qualifiés de "machines de guerres" car ils ont été optimisés pendant plusieurs années par une armée d'ingénieurs. Ils peuvent donc être très efficaces.

Dans tous les cas, la complexité est trop importante pour utiliser ces méthodes sur de grosses instances. On peut alors regarder des heuristiques.

- **Approche gloutonne :** qui fonctionne à peu près de la même façon que celle présentée pour SSP.
- **Recherche locale :** le principe est de partir d'une solution initiale et de tenter de l'améliorer en regardant des solutions voisines jusqu'à obtenir un maximum local. Par exemple, pour explorer des solutions voisines, on peut par exemple échanger la place de deux sommets dans le chemin initial (*2-interchange*), à faire une permutation sur k arêtes (k-*opt*), ou faire plusieurs permutations successives de deux arêtes (*Lin-Kernighan*). Pour trouver une bonne solution, il peut être intéressant de relancer l'algorithme plusieurs fois en variant la solution initiale. *Exemple de 2-opt :* Supposons qu'on ait le chemin contenant deux arêtes $(F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8)$, une solution voisine peut être obtenue en faisant un $2-opt$ sur les arêtes F_2F_3 et F_6F_7. On obtient alors le chemin $(F_1, F_2, F_6, F_5, F_4, F_3, F_7, F_8)$.
- **Programmation linéaire :** une modélisation peut elle aussi être utilisée pour résoudre le problème de façon approchée, on va alors relâcher la contrainte des nombres entiers : en effet trouver la solution à un modèle est polynomial quand les valeurs sont des nombres réels. Après ce relâchement en nombre réels, on essai de trouver une solution en nombres entiers proche de la solution avec les valeurs réelles. Cependant la solution retournée n'est plus exacte. Le solveur Concorde a par exemple réussi à résoudre une instance contenant 65000 sommets en quelques heures.

Enfin, dans la littérature, ce type de problème est beaucoup plus étudié dans une variante symétrique, tandis que la variante utilisée pour l'assemblage est asymétrique. On peut se ramener au cas symétrique en dupliquant les sommets : ainsi un sommet F_i va être décomposé en deux sommets F_i^{in} et F_i^{out} où F_i^{in} va garder les arêtes entrantes vers F_i et F_i^{out} va recevoir les arêtes sortantes de F_i. Voir la Figure 6 pour un exemple.

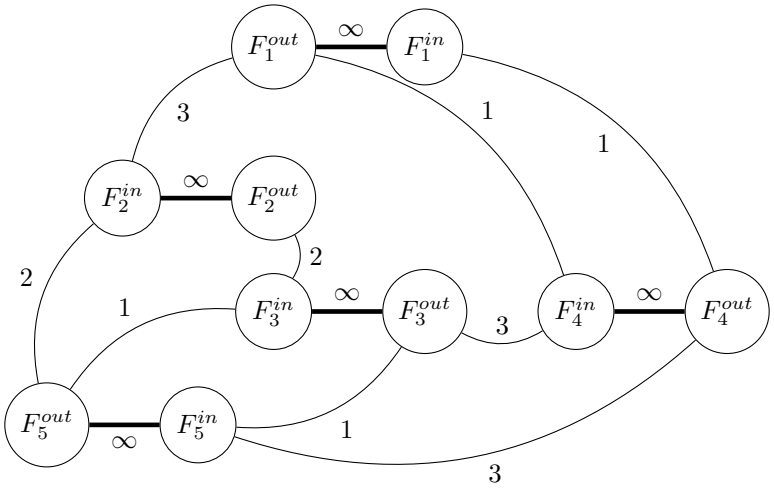


Figure 6: Graphe de chevauchement rendu symétrique.

5.3 Consensus

Dans cette phase utiliser le chemin hamiltonien calculé dans la phase précédente pour établir des consensus sur la séquence originelle. Pour cela, on fait un alignement des mots qui se suivent au sein du chemin hamiltonien. Ainsi séparer le problème de l'assemblage en trois phases permet de s'affranchir de l'information des lectures durant la phase qui nécessite un travail important au niveau du calcul. Cela permet notamment de gagner beaucoup de place en mémoire. Cependant, cela on reste avec un problème NP-difficile à résoudre.

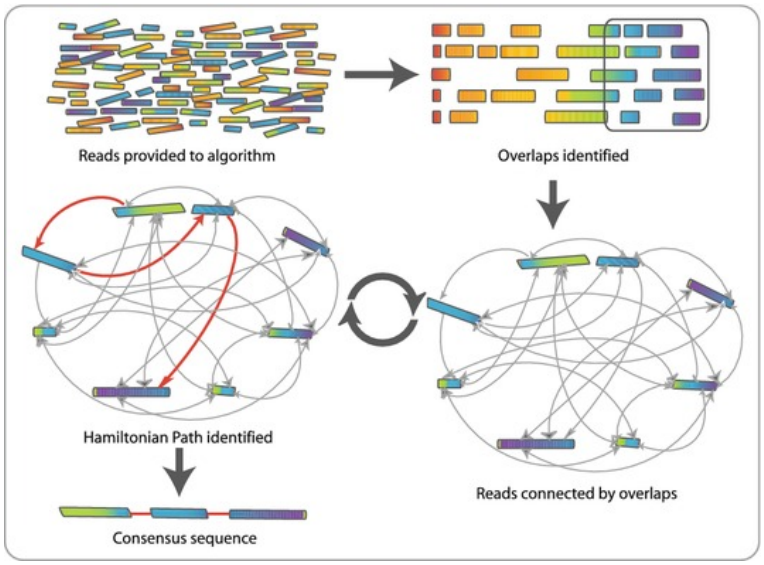


Figure 7: Schéma de la méthode Overlap-Layout-Consensus.
D'après J.Commins et al.,Computational Biology Methods and Their Application to the Comparative Genomics of Endocellular Symbiotic Bacteria of Insects, Bio. Proc. Online. 2009

6 Méthode du chemin eulérien

On présente maintenant une autre approche utilisant un autre type de graphe lié aux lectures : le graphe des k-mers, qui est un cas particulier de graphe de *De Bruijn*. La définition du graphe de De Bruijn $G_{n,L}$ d'ordre L sur l'alphabet $\mathcal{A}$ de taille n est la suivante :

- l'ensemble des sommets de $G_{n,L}$ est l'ensemble de tous les mots de longueurs L dans l'alphabet $\mathcal{A}$, c'est-à-dire que le graphe contient exactement L^n sommets ;
- il existe un arc entre deux mots u et v si le suffixe de longueur $L-1$ de u est égal au préfixe de longueur $L-1$ de v.

Par exemple, construisons le graphe de De Bruijn d'ordre 2 sur l'alphabet $\{0,1\}$. L'ensemble des sommets est égal à $\{00,01,10,11\}$. L'ensemble des arcs est égal à $\{(00,01),(01,10),(01,11),(10,00),(10,01),(11,10)\}$.

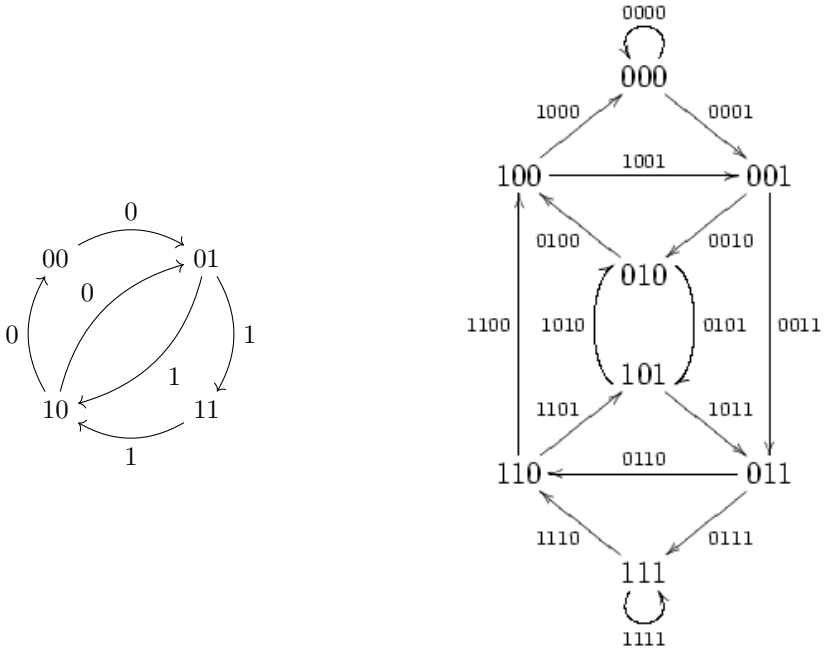


Figure 8: Graphes de De Bruijn d'ordre 2 et 3 sur l'alphabet $\{0,1\}$.

Les graphes de De Bruijn sont *eulériens*, c'est-à-dire qu'on peut parcourir tous les arcs du graphe en utilisant exactement une fois chaque arc, tout en commençant et s'arrêtant ce parcours sur le même sommet. Par exemple, dans le graphe de De Bruin d'ordre 2 sur l'alphabet $\{0,1\}$, un chemin eulérien est donné par $\{(00,01),(01,10),(10,01),(01,11),(11,10),(10,00)\}$.

La version orientée du *Théorème d'Euler* nous indique qu'un graphe orienté fortement connexe est eulérien si et seulement si pour chaque sommet, le degré entrant est égal au degré sortant.

Intuitivement, le théorème se vérifie facilement : si un sommet v a un degré entrant égal à k et un degré sortant égal à $k+\ell$, alors on pourra "entrer" et "sortir" de v k, mais alors k arcs sortants de v ne pourront pas être parcourus.

Pour le problème d'assemblage, on va créer un graphe des k-mers. Pour cela, on va décomposer les lectures en sous-séquences de taille k. Par exemple la lecture `ATTGCTAACTGT` peut être décomposée de la façon suivante en 6-mers: `ATTGCT, TTGCTA,TGCTAA,GCTAAC,CTAACT,TAACTG,AACTGT`. La méthode du chemin eulérien pour l'assemblage est décomposée en plusieurs phases.

1. On choisit un nombre k impair suffisamment petit pour capturer les chevauchements et suffisamment grand pour éviter les répétitions.
2. On nettoie les lectures avec des méthodes statistiques qui permettent de détecter des erreurs de séquençage.
3. On construit le graphe des k-mers. Le stockage des k-mers à proprement parlé est réalisé dans une structure à part pour optimiser l'espace.

4. On nettoie le graphe : l'analyse de la structure permet de détecter des erreurs.
5. Enfin on recherche des chemins eulériens.

On verra certaines de ces phases plus en détails lors des TD. Le calcul d'un chemin eulérien est polynomial, mais pour autant, ça ne rend pas facile la résolution du problème de l'assemblage : en effet la quantité d'informations à stocker peut être énorme. Beaucoup de travail doit être fait en utilisant des outils d'algorithme du texte avancés. De plus gardons bien à l'esprit qu'un algorithme en temps polynomial peut mettre du temps à s'exécuter si la taille de l'entrée est grande.

7 Nettoyer l'assemblage

La plupart des assembleurs effectuent des post-traitements pour réaliser les tâches suivantes.

- Corriger les erreurs de séquençage. Cela peut être fait en observant des erreurs structurelles ou en éliminant les parties peu couvertes par des lectures.

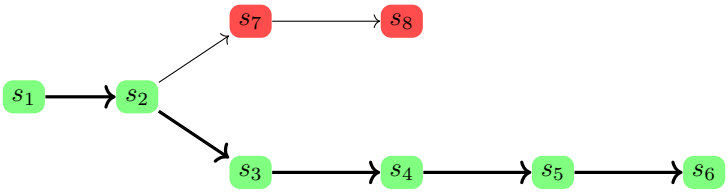


Figure 9: Erreur de séquençage. L'épaisseur du trait est corrélé au nombre de chevauchements passant par ces chemins. Les mots s_7 et s_8 sont probablement issus d'erreurs de séquençage, car peu de chemins les empruntent et ils constituent une branche pendante du graphe.

- Détecter les polymorphismes, c'est-à-dire différentes versions d'un même gène (*allèle*).

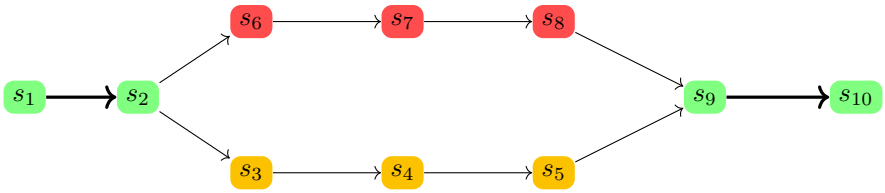


Figure 10: Polymorphisme. Les séquences $s_3s_4s_5$ et $s_6s_7s_8$ sont polymorphes, on peut le voir car les deux séquences $s_1s_2s_3s_4s_5s_9s_{10}$ et $s_1s_2s_6s_7s_8s_9s_{10}$ sont cohérentes.

- Gérer les répétitions.

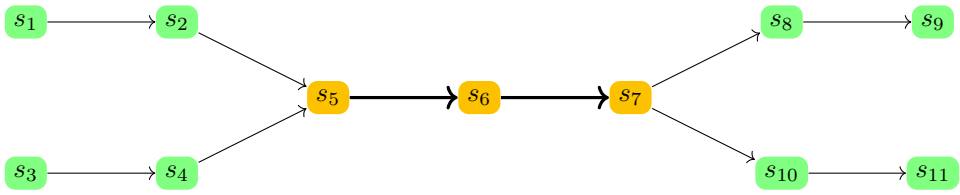


Figure 11: Répétition. La séquence $s_5s_6s_7$ est répétée deux fois dans la séquence originelle car deux chemins passent par elle.

8 Après l'assemblage

Après l'assemblage, toutes les parties de la séquence obtenue ne sont pas forcément correctes car l'information séquencée contient des trous. On ne garde donc que les régions qui ont une couverture importante des lectures issues du

séquençage. Ces régions sont appelées *contigs* et sont généralement de grande taille. Après l'assemblage, il reste à placer ces contigs au sein du génome. La encore, les informations issues du séquençage vont être utilisées pour placer et orienter ces contigs : il s'agit de l'opération d'échafaudage (scaffolding en anglais). Il s'agit là aussi d'un problème NP-difficile.

Figure 12: Échafaudage. On se sert des lectures pairées, qui est une information fournie par le séquençage, pour relier les contigs et connaître leurs sens de lecture (un contig `AATCGAT` peut être également lu `TAGCTAA`). En bas, les flèches représentent les contigs et leurs sens de lecture. En haut, sont dessinés les lectures pairées, c'est-à-dire des lectures issues de la même molécule d'ADN. Plus le nombre de paires partagées par deux contigs est grand, plus la probabilité que ces deux contigs se suivent dans la séquence génomique est importante.

Même après l'échafaudage, de nombreuses opérations restent à faire : il reste toujours des trous, des incohérences ou des ambiguïtés à enlever. Cela est fait "à la main" en laboratoire à l'aide d'expériences supplémentaires. Comme il s'agit d'opérations très coûteuses, à la fois en termes de temps ou d'argent. C'est pourquoi on ne complète que les génomes dits à "haute priorité". Les autres génomes restent à l'état de brouillon dans des bases de données ; ils permettent quand même d'obtenir des informations très importantes d'un point de vue scientifique.