Philippe Leleux

M2R IAICI, **Université Paul Sabatier**

Rapport de stage de recherche

—

Assemblage de génomes à l'aide des réseaux de fonctions de coûts

Tuteur de stage : Simon de Givry

Unité MIAT, **INRA**

Abstract

A mi chemin entre informatique et biologie mon stage de fin d'étude porte sur le problème d'optimisation combinatoire du scaffolding, problème inhérant à l'assemblage de génome.
J'ai effectué ce stage dans l'unité MIAT de l'INRA, sous la tutelle de Simon de Givry. Mes différentes contributions ont porté sur :

- l'évaluation et la visualisation de scaffolds,
- la modélisation de certains problèmes dans le cadre des Problèmes de Satisfaction de Contraintes Pondérées et de la Programmation Linéaire,
- la reprise du processus d'un scaffolder connu, Sopra, pour intégration de ces modèles,
- des expérimentation et comparaisons sur différents scaffolders et génome. et leur comparaison.

A terme, j'ai pu candidater pour obtenir une bourse de thèse sur le sujet avec l'école doctorale MITT.

Monday 15th September, 2014

Contents

1 Introduction

1.1 Unité MIAT de l'INRA

L'**INRA (Institut National de la Recherche Agronomique)** est le premier institut de recherche public européen et le deuxième dans le monde pour ses publications en sciences agricoles, sciences des plantes et de l'animal. Il rassemble 8478 agents titulaires dont 1800 chercheurs répartis sur 19 centres de recherche. Son ambition est, dans une perspective mondiale, de contribuer à assurer une alimentation saine et de qualité, une agriculture compétitive et durable ainsi qu'un environnement préservé et valorisé.

Le centre de Toulouse rassemble plus de 850 chercheurs, ingénieurs et techniciens INRA, dont 600 titulaires. Ce centre représente environ 10% des publications et près de 12% des brevets de l'institut autour de 7 axes de recherche. Lieu d'activités pluridisciplinaires, ce centre m'aura permis d'approcher deux mondes différents en alliant la biologie inhérante à l'institut ainsi que l'informatique, spécialité de l'**unité Mathématiques et Informatique Appliquées de Toulouse (MIAT)**. Cette unité compte sur l'activité de 3 plateformes :

- **Plateforme Bioinformatique :** centrée sur l'analyse de séquences et faisant partie du GIS Génotoul (plateforme technologique mettant à disposition un cluster de calcul [1]),
- **Plateforme RECORD :** offre un cadre et des outils informatiques communs aux modélisateurs de différentes disciplines,
- **Plateforme SIGENAE :** regroupement d'ingénieurs en bio-informatique qui accompagnent les biologistes dans le traitement de données à haut débit.

L'unité est réunie en 2 équipes de recherche qui collaborent avec ces plateformes :

- **MAD (Modélisation des Agro-écosystèmes et Décision) :** modélisation des systèmes complexes dans les champs de l'agriculture, de l'environnement et de l'analyse des risques alimentaires et des procédés industriels (Plateforme Record),
- **SaAB (Statistique et Algorithmique pour la Biologie) :** bioinformatique, au sens de l'ensemble des méthodes relevant des mathématiques et de l'informatique appliquées à l'exploitation des données de génomique et de post génomique (Plateforme Bioinformatique).

1.2 Contexte

C'est dans l'équipe **SaAb** de l'**unité MIAT** que j'ai été intégré sous la tutelle de **Simon de Givry**. J'ai eu la chance de pouvoir utiliser pour mes calculs le **cluster** de la **plateforme bioinformatique** et ses 34 noeuds de 4*12 coeurs de 32 GB de ram. Mais ce que ce stage m'aura surtout permis c'est de cottoyer des acteurs de différents domaines. J'ai ainsi pu faire la connaissance de chercheurs titulaires, d'ingénieurs recherche, de doctorants et de post-doctorants. Ils ont pu m'apporter leur savoir que ce soit dans le domaine de la biologie (où mes connaissances préalables remontaient à la fin du lycée) ou de l'informatique sans oublier la recherche en général.

Je voudrais particulièrement remercier mon tuteur, ainsi que **Thomas Schiex** et **David Alouche** pour leur aide tout au long du stage. Sans oublier **Christophe Klopp** (responsable de la plateforme Bioinfo) et **Matthias Zytnicki** pour leur expertise en bio-informatique et nottamment dans l'analyse de séquences.

[1] `http://bioinfo.genotoul.fr/`

J'ai également eu la chance de participer à une collaboration avec une **équipe de chercheurs du LIRMM[2], laboratoire de Montpellier** (que je désignerai seulement comme **le LIRMM** dans la suite). Dans cette équipe, je voudrais remercier **Annie Château**, **Rémi Colleta** et **Nicolas Briot**. C'est au travers de cette collaboration que j'ai pu dès le 2ème mois de mon stage aller présenter des expérimentations effectuées sur différents scaffoldeurs (Voir section 4.3). Avec eux nous avons également écrit, lors du 4ème mois, un article pour le **workshop CP'2014** de Lyon. Nous sommes allés le présenter durant la conférence conjointement avec Simon de Givry. En raison de l'expérience que j'avais gagné dans la manipulation et l'évaluation de scaffoldeurs, j'ai été chargé lors de la rédaction d'apporter tous les résultats de la partie expérimentale de l'article. Cette équipe de Montpellier a en effet développé, avec notre aide, leur propre scaffoldeur, **scaftools** [1], dans un cadre de programmation linéaire.

1.2.1 Sujet

Les plates-formes de séquençage de très haut débit produisent d'énormes quantités de petites lectures (< 100bp) à chaque lancement. Alors que ces petites lectures sont adéquates pour des opérations de re-séquençage, l'assemblage de novo de génomes reste un challenge. Ces limitations pourraient être partiellement surmontées en utilisant la technologie des paires de lectures qui fournit des paires de courtes séquences séparées par une distance connue dans le génome.

Mon stage constitue le commencement des travaux de l'équipe de Simon de Givry dans le domaine même si l'approche adoptée - celle des réseaux de fonctions de coût - paraît naturelle et nous le verrons dans la suite. Nous avons donc conjointement étudié les tenant et aboutissant du sujet en détails, détails que je présenterai en section 3. L'objet du stage est l'étude de différentes modélisations du problème de scaffolding et mise en œuvre avec reformulation dans le cadre des problèmes de satisfaction de contraintes pondérées (WCSP). Mon travail a également consisté en l'étude de différentes métriques d'évaluation des différents scaffoldeurs qu'il faudra comparer. Des outils de visualisation du génome mériteraient d'être adaptés à la visualisation de scaffolds comme il n'existe pas d'outil dédié à ce jour.

1.2.2 Déroulement

Le stage s'est déroulé en plusieurs "phases" :

1. Bibliographie,
2. Expérimentations,
3. Déchiffrage Sopra,
4. Visualisation de scaffoldeurs,
5. Préparation candidatures de thèses,
6. Comparaison/Evaluation de scaffoldeurs,
7. Collaboration avec le LIRMM pour la rédaction d'un article,
8. Reprise du protocole de Sopra en intégrant nos modèles.

Durant ces phases, se sont passées différentes actions discrètes :

- 25/03 : Remise d'un **état de l'art** sur le sujet après dix jours de stage pour l'UE culture scientifique du M2R IAICI,
- **Collaboration avec le LIRMM :**

[2] `https://www.lirmm.fr/`

  - 4/04 : Réunion à Montpellier pour qu'ils nous présente leur modèle pseudo-booléen. Simon leur a présenté les avantages et inconvénients du protocole Sopra et moi un rapport sur une phase d'expérimentation préalable. Cette expérimentation a consisté à lancer plusieurs scaffoleurs sur plusieurs génomes pour en comparer les performances. Je présenterai plus en détails ce travail dans la section 4.3,
  - mois de juin puis mi-août : **Participation à l'écriture d'un article** pour le workshop CP'2014 dans lequel je me suis occupé de tous les résultats expérimentaux ainsi que de plusieurs relectures et corrections. J'en suis co-auteur,
  - 8/09 au 12/09 : **Participation au workshop** en présentant l'article à la conférence conjointement avec Simon de Givry.

- 03/06 (après plusieurs jours d'entraînement) : Oral pour la **candidature à une bourse de thèse avec l'école doctorale MITT** qui aurait pour directeur de thèse Simon de Givry sur le sujet du stage,
- 19/06 : Présentation de l'avancée du travail du stage au reste de l'unité à l'occasion de l'annuelle "journée des stagiaires".
- 25/06 : **Entretien pour la proposition de thèse de Thomas Schiex** à l'INRA et l'INSA sur le thème du "Computational Protein Design".

L'organisation avec le reste des personnes que j'ai cotoyé se déroulait ainsi :

- **Réunion avec mon tuteur** au moins une fois par semaine afin de faire un point sur mon avancement et de décider des prochaines actions.
- **Rencontre avec le tuteur technique de l'ENSEEIHT**, Daniel Ruiz en début du deuxième mois de stage.
- Quelques rencontres avec Christophe Klopp, David Alouche et Matthias Zytnicki qui m'ont permis de ne jamais rester coincé trop longtemps grâce à leur expertise en bioinformatique. Ils ont su se montrer toujours disponibles et accueillants.
- J'ai tenu à jour une **documentation** de mon stage avec planning, mails, compte-rendus de réunions, rapports sur certains travaux (Expérimentations multiples, étude de Sopra, papier sur le fonctionnement de circos, rapport de mi-stage, slides de présentations...)

1.2.3 Objectifs et enjeux

L'objectif du stage a beaucoup évolué au fur et mesure des mois et beaucoup d'éléments imprévus se sont ajoutés. Pour résumer, les objectifs du stage étaient les suivant :

- Découvrir le problème du scaffolding en profondeur,
- Etudier et comparer différents scaffoldeurs,
- Pour cela, étudier la comparaison/visualisation entre scaffoldeurs,
- Etudier différentes modélisations du problème et les appliquer,
- Reprendre le protocole implanté par Sopra en intégrant les modélisations étudiées grâce à python et la plateforme multi-paradigme Numberjack.

Les enjeux de ce stage sont clairs : pouvoir à terme intégrer l'expertise du laboratoire en matière d'optimisation combinatoire et de bioinformatique dans le domaine du scaffolding. A titre plus personnel, ce stage me permet d'élargir ma formation en informatique et intelligence artificielle pour acquérir une deuxième compétence en biologie. C'est pourquoi je voudrais pouvoir poursuivre avec une thèse dans le domaine. J'ai ainsi postulé à deux thèses:

- Avec l'école doctorale MITT, sur le sujet "Assemblage de génomes à l'aide de données incertaines et hétérogènes" et sous la tutelle de Simon de Givry. J'ai été classé 19ème sur 26 et suis toujours en attente d'une réponse (le processus peut aller jusqu'en automne).
- En collaboration INRA-INSA, sur le thème des "Computational Protein Design" et sous la tutelle de Thomas Schiex et Sophie Barbe. J'ai été classé second suite aux entretiens[FI et n'ai donc pas obtenu la bourse.

2 Contexte Méthodologie

2.1 CSP

2.1.1 Les CSPs

Les problèmes de **Satisfaction de contraintes (CSPs pour Constraint Satisfaction Problems)** sont des problèmes mathématiques définis comme un ensemble d'objets qui doivent (absolument) satisfaire un ensemble de contraintes (ou limitations). Les CSPs sont sujets à d'intenses recherche en intelligence artificielle ou en recherche opérationelle comme la régularité de leur formulation offre une base commune pour analyser et résoudre des problèmes très variés. Les CSPs ont souvent une très grande complexité et nécessitent une combinaison d'heuristiques et de méthodes de recherche combinatoire pour être résolus en un temps raisonnable. Parmi les CSPs, on compte par exemple, la coloration de carte (ou de graphe), le Sudoku, l'allocation de ressources...

2.1.2 Définition formelle

Considérons un problème dans le cadre des problèmes de satisfaction de contraintes (CSP). Ce problème est défini par un triplet **‹X, D, C›** avec :

- X un ensemble de variables,
- D l'ensemble des domaines finis pour les variables,
- C un ensemble de contraintes.

Chacune de ces contraintes porte sur un sous-ensemble des variables et n'autorise qu'une partie des combinaisons possibles de valeurs pour ces variables.

Soit $t_j \subset X$ un sous-ensemble de k variables, R_j une relation d'arité k sur le sous-ensemble correspondant d_j : une contrainte est alors (t_j, R_j).
Une *évaluation* des variables est une fonction d'un ensemble des variables dans le sous-ensemble correspondant de domaines. Cette évaluation *satisfait* une contrainte si l'assignation des valeur de t_j satisfait la relation R_j. Une évaluation est :

- *Consistante* si elle ne viole aucune contrainte,
- *Complète* si elle inclut toutes les variables,
- *Une solution* si elle est *consistante* et *complète*, elle résout alors le problème.

Ces problèmes peuvent être réprésentés sous la forme d'un **graphe de contraintes** (voir Figure 1) : graphe non-orienté où chaque noeud représente une variable et il existe une arête entre deux noeuds si et seulement si il existe une contrainte qui relie les deux variables.

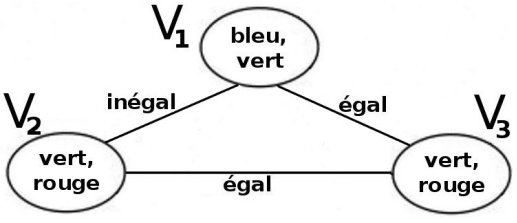

Figure 1: Modélisation d'un problème de coloration de graphe à 3 variable en CSP. Chaque arête correspond à une contrainte d'égalité ou d'inégalité.

2.1.3 Résolution

Les CSPs sur des domaines finis sont le plus souvent résolus à l'aide d'une forme de recherche. Les techniques les plus utilisées sont des variantes du **backtracking**, de la **propagation de contraintes**, et de la **recherche locale**.

Backtracking (BT) :
Le backtracking est un *algorithme récursif*. Il maintient une affectation partielle des variables:

1. Initialement, aucune variable n'est assignée,
2. A chaque étape, on choisit une variable et toutes les valeurs possibles lui sont assignées une à une,
3. Pour chaque valeur, la cohérence de l'affectation partielle avec les contraintes est vérifiée et en cas d'incohérence, un appel récursif est effectué,
4. Lorsque toutes les valeurs ont été essayées, l'algorithme "backtrack" c'est-à-dire revient en arrière.

Dans cet algorithme de backtracking basique, la cohérence est définie comme la satisfaction de toutes les contraintes portant seulement sur des variables déjà affectées. Plusieurs versions de l'algorithme de backtracking existent telles que **Backmarking**, **Backjumping**, **Constraint** ou **Look-ahead**.

Propagation de contraintes :
Les techniques de propagation de contraintes sont des méthodes utilisées pour modifier un CSP en forçant une forme de cohérence locale (condition liée à la cohérence d'un groupe de variables et/ou de contraintes). Elles transforment ainsi un problème en **un autre équivalent qui est (généralement) plus simple à résoudre**. Les formes les plus connues et utilisées des cohérences locales sont :

- L'arc cohérence,
- L'hyper-arc cohérence,
- La cohérence de chemin.

Recherche locale :
Les méthodes de recherche locales sont des algorithmes de satisfiabilité *incomplets* (elles peuvent échouer même si le problème est satisfiable). Ils améliorent itérativement une assignation complète des variables. A chaque étape, les valeurs d'un petit nombre de variables sont changées, avec pour but d'augmenter le nombre de contraintes satisfaites par cette affectation. L'agorithme **Min-Conflit** est un algorithme de recherche locale spécifique aux CSPs et basé sur ce principe.

Dans le cas où aucune affectation ne peut satisfaire toutes les contraintes, il existe différents formalismes qui cherchent à obtenir la "meilleure" solution (définition de laquelle peut varier d'un formalisme à l'autre). Parmi ces formalismes, on retrouve par exemple le cadre **Max-SAT**, qui cherche à trouver une solution maximisant le nombre de contraintes satisfaites. Ce cadre est par exemple exploré par le LIRMM pour le scaffolding. Mais le cadre que nous avons choisi, et dont certains membres de l'unité MIAT sont des experts, est le cadre **WCSP**.

2.2 WCSP

2.2.1 Les WCSPs

Les **problèmes de Satisfaction de contraintes pondérés (WCSPs pour Weighted CSPs)**, comme présenté dans [2], sont une **extension des CSPs** permettant de résoudre des problèmes sous contraintes qui sont *en général insatisfiables*. On introduit pour cela un procédé permettant d'exprimer des **préférences sur la solution** du problème. De même que pour les CSPs, ces problèmes nécessitent donc souvent heuristiques et méthodes de recherche combinatoire. Parmi les applications, tous les CSPs insatisfiables ainsi que le problème des Bandwidth et le Computational Protein Design...

2.2.2 Définition formelle

Considérons un problème dans le cadre des WCSPs. Ce problème est défini par un **triplet ‹X, D, F›** avec :

- X un ensemble de variables,
- D l'ensemble des domaines finis pour chaque variable,
- F un ensemble de fonctions. Chacune de ces fonctions est définie sur un ensemble des variables (le nombre de ces variables s'appelle l'*arité*, dans $\mathbb{N}$) et associe à chaque affectation de celles-ci un entier que l'on appellera *coût*. En particulier, la fonction d'arité 0 correspond à un coût commun à toutes les affectations.

Le *coût d'une affectation* sera alors la somme de toutes les fonctions appliquées à l'affection, où la somme utilisée est une *somme bornée*, la borne correspondant au *coût d'une affectation interdite*. Le problème en WCSPs sera alors de chercher la (ou les) *affectation complète de coût minimum* qui sera une solution du problème.

On peut ainsi exprimer dans une solution des préférences sur les variables au travers des coûts sans pour autant contraindre le problème de manière absolue.

Ces problèmes, comme les CSPs, peuvent être réprésentés sous la forme d'un **graphe de contraintes pondérées (ou réseau de fontions de coûts)** (voir Figure 2) : graphe non orienté où chaque noeud représente une variable et il existe une arête entre deux noeuds si et seulement si il existe une fonction de coût qui relie les deux variables.

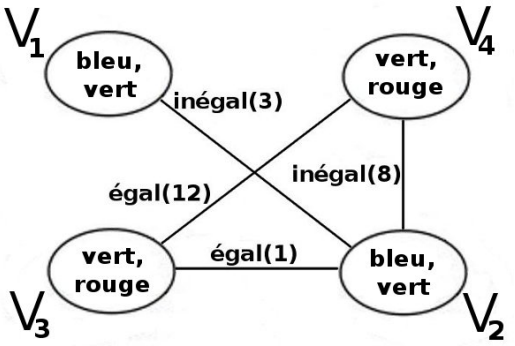

Figure 2: Modélisation d'un problème quelconque en WCSP : X = {V_1, V_2, V_3, V_4} et D = {{1, 3}, {1, 3}, {2, 3}, {2, 3}}. F est composé de fonctions d'arité 2 correspondantes aux propriétés d'égalité/inégalité des arêtes. Pour chaque fonction, le coût est celui entre parenthèses lorsque la propriété est violée, 0 sinon. Une solution optimale est (3, 1, 2, 2) de coût 1.

2.2.3 Résolution

Pour les WCSPs, les techniques se rapprochent de celles utilisées en CSPs et consistent toutes en une **recherche dans l'ensemble des affectations possibles**. Plusieurs techniques d'optimisation sont alors envisageables, notamment un **algorithme glouton** ou l'**élimination de variable** en adaptant les méthodes prévues pour les CSPs. Cependant, c'est une autre méthode exacte qui est plus classiquement utilisée : l'**algorithme de séparation et évaluation (branch and bound) avec parcours en profondeur d'abord (Depth-First Branch and Bound ou DFBB)** combiné avec une **décomposition en arbre de clusters de calcul (Branch Tree Decomposition ou BTD)**.

Cette méthode est représentée par un arbre de recherche, où chaque branche représente une instanciation du problème et chaque nœud est un sous-problème, un cluster de variables (voir Figure 3).

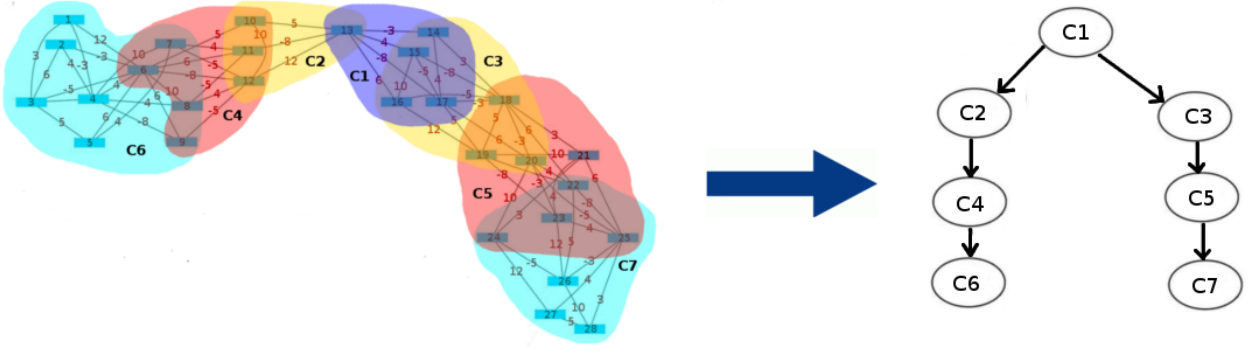

Figure 3: Transformation d'un problème WCSP en arbre de clusters. Chaque cluster est composé par les variables dans la couleur associée (par exemple l'ensemble bleu pour le cluster C1).

DFBB a une complexité de O(d^n) où d est la taille du plus gros sous-problème. A l'instar de l'agorithme de backtrack (BT) pour les CSPs, cet algorithme énumére les valeurs possibles des variables récursivement, en remontant à la variable précédemment affectée **dès qu'une estimation du coût de n'importe quelle extension complète de l'affectation courante (i.e. le minorant du sous-problème courant) dépasse le coût de la meilleure solution déjà visitée k (i.e. le majorant du problème d'origine, initialement**

supposé de coût infini). Meilleur est le minorant du problème courant, plus petit sera l'espace parcouru lors de la recherche suivante. Un minorant trivial consiste à sommer le coût des fonctions dont les variables sont affectées. Il existe des propriétés dites de **cohérences souples** qui permettent de calculer des minorants plus précis avec des algorithmes plus sophistiqués. Ces minorants seront alors généralement stockés dans la fonction d'arité nulle, commune à toutes les affectations. Dans le cas des **fonction de coûts binaires** (d'arité 2), selon [3], les cohérences souples sont :

Cohérence locale	Complexité		Classes polynomiales
	En temps	En espace	
NC*	O(nd)	O(nd)	-
BAC∅	$O(n^2d^3)$	O(e)	-
AC*	$O(n^2d^2 + ed3)$	O(ed)	-
DAC	$O(ed^2)$	O(ed)	Arbre
FDAC	$O(end^3)$	O(ed)	Arbre
EDAC*	$O(ed^2max(nd, k))$	O(ed)	Arbre
VACϵ	$O(\frac{ed^2k}{\epsilon})$	O(ed)	Arbre, fsm
OSAC	poly(ed + n)	$poly(ed^2 + nd)$	Arbre, fsm

Les points cruciaux de la résolution des WCSPs sont ainsi la **décomposition en arbres de clusters de calcul** et **le calcul des minorants des sous-problèmes**. Comme nous le verrons dans la section 3, cette modélisation paraît très naturelle pour le problème qui nous intéresse : **le scaffolding**.

2.3 Optimisation

Un **problème d'optimisation combinatoire (ou d'optimisation discrète)** consiste à trouver dans un ensemble discret un parmi les meilleurs sous-ensembles (ou solutions) réalisables, la notion de *meilleure solution* étant définie par une *fonction objectif*. Formellement :

- un ensemble discret N,
- une fonction d'ensemble f : $2^N \to \mathbb{R}$ (fonction objectif),
- un ensemble $\mathcal{R}$ de sous-ensembles de N (les éléments sont appelés **solutions réalisables**).

Le problème combinatoire consiste alors à déterminer : $\max_{S \subseteq N}\{f(S) : S \in \mathcal{R}\}$.

Il existe différents formalismes permettant de modéliser ce genre de problèmes. On voit d'ailleurs assez naturellement que les WCSPs en font partie (ainsi qu'à l'extrême les CSPs). La suite présente un bref aperçu des autres formalismes que j'ai pu utiliser durant ce stage.

2.3.1 Programmation linéaire

La programmation linéaire (ou optimisation linéaire abrégée LP pour Linear Programming) est une méthode permettant de trouver la meilleure solution à un modèle mathématiques pouvant être représenté par une **relaton linéaire**. De manière plus formelle, LP est une technique d'**optimisation d'une fonction objectif linéaire avec des contraintes d'égalités et d'inégalités linéaires**.

Les problèmes peuvent être exprimés de manière canonique ainsi :

$$\begin{aligned} &\text{maximize} && \mathbf{c}^{\mathrm{T}}\mathbf{x} && (1)\\ &\text{subject to} && A\mathbf{x} \leq \mathbf{b} && (2)\\ &\text{and} && \mathbf{x} \geq \mathbf{0} && (3) \end{aligned}$$

avec :

- x le vecteur des variables à déterminer,
- c et b des vecteurs de coefficients connus,
- A une matrice de coefficients connus,
- $(\cdot)^{\mathrm{T}}$ la transposée de matrice.

La programmation linéaire en nombres entiers (ILP pour Integer Linear Programming) est un cas spécial de LP où les domaines des variables à déterminer sont entiers. On ajoute ainsi la contrainte $\mathbf{x} \in \mathbb{Z}^n$

2.3.2 Programmation quadratique

La programmation quadratique (QP pour Quadratic Programming) est le problème d'**optimiser (minimiser ou maximiser) une fonction quadratique sur plusieurs variables avec des contraintes linéaires sur ces variables**.

Le problème quadratique peut être formulé ainsi :

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} && (4)\\ &\text{subject to} && A\mathbf{x} \leq \mathbf{b}(inequalityconstraint) && (5)\\ &\text{and} && E\mathbf{x} = \mathbf{d}(equalityconstraint) && (6) \end{aligned}$$

avec :

- $\mathrm{x} \in \mathbb{R}^n$,
- x, c vecteurs colonnes à n éléments,
- Q une matrice symétrique nxn,
- $A\mathbf{x} \leq \mathbf{b}$ signifie $A\mathbf{x}$ est inférieur ou égal à la valeur correspondante dans le vecteur $\mathbf{b}$,
- $(\cdot)^{\mathrm{T}}$ la transposée de matrice.

2.3.3 Programmation mixte

La programmation linéaire mixte (MIP pour Mixte Integer Programming) est une extension de la LP où quelques variables seulement sont entières alors que les autres sont réelles.

2.3.4 Programmation dynamique

La programmation dynamique est une méthode permettant de **résoudre des problèmes complexes en les découpant en sous-problèmes plus simples**. C'est une méthode applicable lorsque le problème montre des **propriétés de chevauchement entre sous problèmes** et des **sous-structures optimales**. Lorsqu'on peut l'appliquer, cette méthode est beaucoup plus rapide qu'une méthode naïve.

L'idée derrière cette méthode est simple : pour résoudre un problème donné, on a besoin de résoudre différents sous-problèmes puis combiner les solutions partielles pour atteindre une solution totale. Souvent avec une méthode naïve, chaque sous-problème est résolu plusieurs fois tandis qu'en programmation dynamique, on cherche à ne résoudre chaque sous-problème qu'une fois en mémorisant la solution. Un algorithme de programmation dynamique va examiner les sous-problèmes précédemment résolus et combiner leur solution pour obtenir la meilleure solution du problème. Cette méthode s'avère très utile lorsque le nombre de problèmes qui se répètent croît exponentiellement en la taille de l'entrée.

3 Contexte biologique

3.1 La génomique

Séquencer, assembler et analyser le génome des espèces est un problème majeur de ces dernières décénnies. Tout a commencé avec le HGP.

"Le HGP ou Projet Génome Humain est un projet de recherche scientifique international ayant pour but de déterminer la séquence des paires de bases qui constitue l'ADN humain, et d'identifier et cataloguer l'ensemble des gènes du génome humain d'un point de vue à la fois physique et fonctionnel. Il constitue à ce jour le plus grand projet biologique collaboratif du monde."

Wikipedia - Human Genome Project, traduit de l'anglais

3.1.1 ADN et génome

Alors que d'un point de vue biologique, l'ADN est une grande molécule composée d'unités répétées (des nucléotides), d'un point de vue informatique, l'ADN n'est autre qu'une grande chaîne de caractères composée de 4 lettres : A, C, T, G. Ces caractères vont par deux (A avec T et C avec G), on parle alors de **paire de base** ou **bp**.

Le génome est l'information qui peut être extraite de l'ADN, par exemple les gènes, les variations entre individus, les variations entre espèces. Connaître le génome d'une espèce est important que ce soit en médecine ou en agronomie. Le génome de chaque individu va être d'une importance grandissante dans le futur au vu des applications potentielles de la médecine personnalisée (customisation des soins pour chaque individu).

3.1.2 Séquençage du génome

Séquencer le génome consiste à passer de la molécule d'ADN à une chaîne de caractère, l'objet informatique (voir Figure 4). Un séquenceur prend en entrée des séquences d'ADN et retourne en sortie des fichiers texte contenant les séquences.

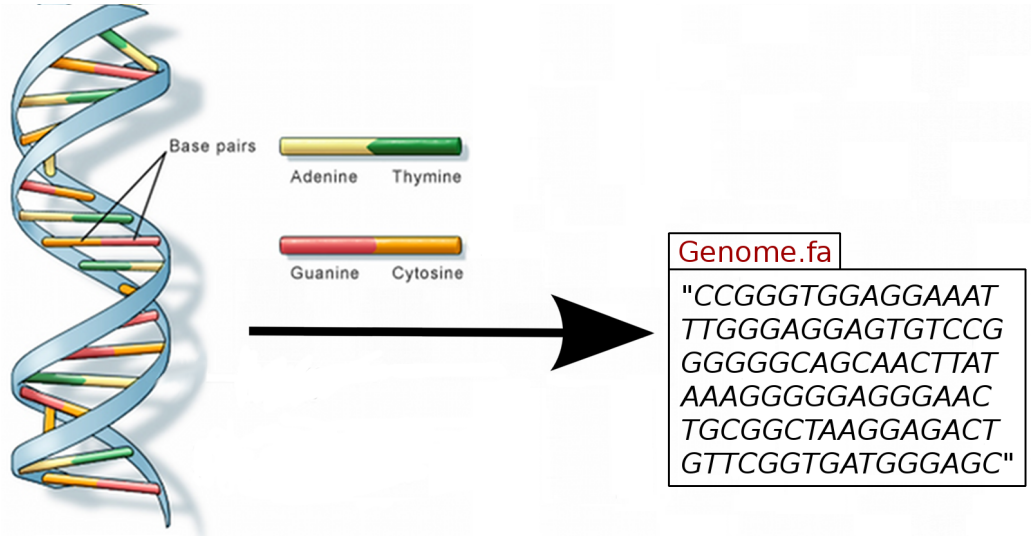

Figure 4: Aspect biologique vs. aspect informatique de l'ADN

3.1.3 Assemblage de séquences

L'assemblage de séquences consiste à aligner et fusionner des fragments d'une chaîne d'ADN plus longue (créées lors de l'étape de séquençage) afin de reconstruire le génome ou la séquence de départ. Cette étape est nécessaire comme la technologie de séquençage de l'ADN ne peut pas obtenir le génome entier en une seule lecture, mais permet d'obtenir des séquences courtes de longueur dépendant de la technologie utilisée.
Voici une idée simple de ce qu'est l'assemblage:
"Le problème de l'assemblage de séquence se rapporte au problème de prendre beaucoup de copies d'un livre, de toutes les passer au travers d'une broyeuse de documents avec différentes tailles et de remettre les morceaux

ensemble juste en les regardant. Outre l'évidente difficulté de la tâche, il existe d'autres problèmes en pratique : le génome de départ peut avoir beaucoup de parties répétées, et quelques fragments peuvent avoir été modifiés durant la phase de broyage occasionant des typos. Des fragments d'un autre "livre" pourraient également s'être mélangés à la mixture et d'autres pourraient être complètement illisibles."

Rayan Chikhi [4] - *Computational Methods for de novo Assembly of Next-Generation Genome Sequencing Data, traduit de l'anglais*

La question naturelle qui se pose est : **est-ce possible de retrouver la séquence originale à partir des fragments donnés en sortie du séquenceur ?** Si la machine retournait une seule copie de la séquence originale (chaque nucléotide est lu exactement une fois), la tâche serait impossible mais, si chaque fragment du génome est lu plusieurs fois, ça devient possible.

3.1.4 Lectures et autre vocabulaire

Il y a beaucoup de vocabulaire à apprendre pour être capable d'apréhender les problèmes de séquençage et d'assemblage du génome. **Matériel :**
Pour un séquençage et un assemblage du génome réussi, nous avons besoin de :

- Fragments de l'ADN (aussi appelés extraits, morceaux ou séquences),
- Une machine de séquençage (ou séquenceur),
- Une capacité importante de calcul et de stockage de données,
- Patience.

Séquenceur :
Un séquenceur de n'importe quel modèle fera l'affaire, même si nous travaillerons plus avec la technologie **Illumina**. Nous allons donner à ce séquenceur les fragments d'ADN en entrée. Le traitement appliqué à ces fragments diffèrera d'un séquenceur à l'autre avec différentes techniques utilisées etc... (voir annexe A pour plus de détails). Concentrons-nous sur les principes de base pour le moment.
Le séquenceur prend les fragments d'ADN et déterminera la séquence de nucléotides de chacune. A partir de là, nous désignerons les fragments provenant du séquenceur comme des lectures, comme c'est le terme le plus largement utilisé. Ces lectures sont de différentes longueur d'un séquenceur à l'autre allant de 25bp à 15000bp. Parfois, et c'est d'autant plus le cas avec les techniques les plus récentes, le séquenceur fera deux lectures sur une séquence d'ADN. C'est ce que l'on appelle des **paires de lecture**. Ces lectures sont faites de manière à ce que l'on puisse connaître leur longueur et la distance qui les sépare, donnant ainsi encore plus d'information pour la prochaine étape, l'assemblage. De plus, comme on connaît un sens sur le génome, ces paires nous donne une information de précédence entre parties du génome. Quand les paires de lecture sont créées, le fragment composé des deux lectures et de l'espace qui les sépare est appelé **insert**. Si l'insert est long (entre 1000bp et 40000bp), les paires de lecture sont appelées **mate-pairs**. Dans le cas contraire, si l'insert est court (pas plus de 500bp), elles sont appelées **paired-ends**. Voir Figure 5 pour une illustration des paires de lecture.

Sequençage shotgun :
La plupart des approches de séquençage utilisent une étape de clonage in vitro pour amplifier les molécules d'ADN individuelles, comme les méthodes de détection moléculaires ne sont pas assez sensibles pour le séquençage de molécules seules. Le séquençage shotgun est une méthode de séquençage faite pour analyser de larges séquences d'ADN ($>$ 1000bp à un génome complet). l'ADN est coupé aléatoirement (shotgun) en de nombreux petits segments, qui sont séquencés
pour obtenir des lectures. Après avoir séquencé des fragments individuels, les séquences peuvent être assemblées.

Assemblage :
Le séquenceur nous a fourni en sortie un fichier texte contenant la séquence déterminée de A, C, G, T et N (nucléotides non-déterminés) aussi bien que des informations concernant la fiabilité de chaque nucléotide.
Nous allons maintenant assembler toutes les lectures. La phase d'assemblage est souvent séparée en 2 phases :

1. D'abord, les lectures sont examinées pour essayer de les mettre en ensemble, en se servant surtout des chevauchements entre séquences qui seront alors fusionnées. Par example si on a les séquences "ACGT", "TTGA" et "CGTT", il semble logique de mettre la prémière et la deuxième séquence ensemble pour formet la séquence "ACGTT". A partir de cette phase, on obtient de plus grandes séquences que nous appellerons **contigs**.

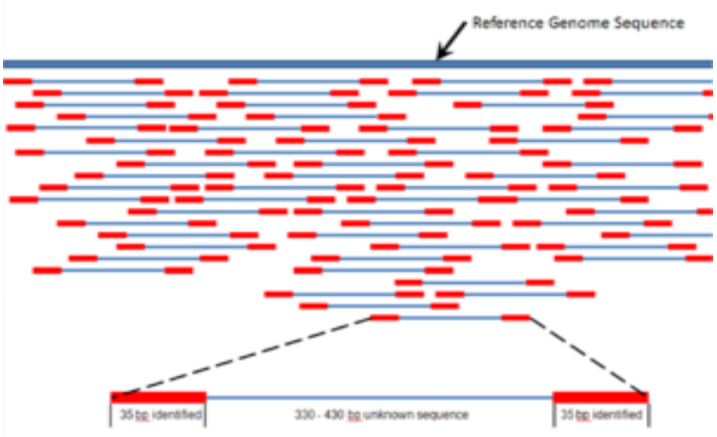

Figure 5: Paires de lecture

2. La seconde phase, le **scaffolding**, consiste à déterminer l'ordre et l'orientation de ces contigs ainsi que la taille de l'espace qui les sépare (si jamais on ne peut déterminer ce qu'il y a à cet endroit), en utilisant l'information des paires de lecture. C'est cette phase qui m'intéresse pendant le stage. A partir de cette phase, on obtient des séquences de contigs quelque fois séparées par des espaces, des gaps, de taille approximativement connue.

Nous avons finalement placé toutes les lectures pour qu'elles nous donne le génome d'origine avec la plus grande précision possible. **On voit que même à la fin de l'assemblage, on n'obtiendra pas forcément la séquence d'ADN complète et en réalité, aucun génome génome n'a jamais été séquencé dans son intégralité avec 100% de certitude (pas même celui de l'humain).** Voir la Figure 6 pour un résumé du processus complet du séquençage "shotgun" à la séquence complètement déterminée.

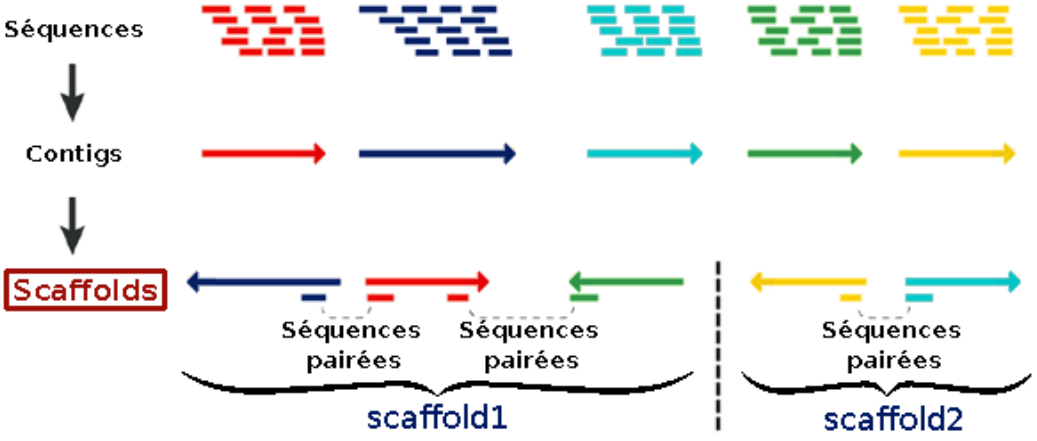

Figure 6: Paires de lectures

Assemblage de novo : L'**Assemblage de novo** se rapporte à l'assemblage d'ADN sans aucune connaissance préalable sur le contenu. C'est le challenge posé à l'assemblage et qui ne connaît pas encore réellement de solution.

3.2 Assemblage de séquences

3.2.1 Principe

L'assemblage de séquence est le processus consistant à aligner, orienter et fusionner les fragments d'ADN obtenus durant la phase de séquençage. Cette étape est nécessaire parce que le séquençage d'ADN ne peut pas retourner en sortie le génome complet en une lecture. Au ieu de cela, on obtient de petites séquences de 20bp à 40000bp (selon la technologie utilisée) en utilisant souvent des fragments venant de séquençage shotgun.
A partir des lectures de la phase de séquençage, l'assembleur va essayer de mettre les lectures ensemble en se basant sur leur chevauchement. L'assemblage est généralement fait en 2 étapes :

- D'abord, les fragments sont assemblés en utilisant le chevauchement des lectures pour construire de plus grandes séquences complètes appelées contigs. L'information utilisée est ainsi généralement limitée aux lectures simple (sans l'information de paire).
- Ensuite vient la phase de scaffolding : les contigs sont ordonnés, c'est à dire qu'on leur donne une orientation, un ordre et que l'on estime la taille des espaces entre ces contigs. chacune de ces séquence est appelée scaffold.

Dans ces deux phases, les séquences (respectivement contigs) sont soit alignés en utilisant un algorithme glouton, soit (plus souvent) mises dans un graphe. Pour l'assemblage en contigs, une séquence valide sera alors un chemin dans le graphe et un scaffold une partie connexe du graphe. Nous examinerons les différentes sortes de graphe utilisées dans les prochaines sections. Pendant le scaffolding, nous avons une séquence de contigs (quelque fois avec des trous) et alors se pose une question : cette séquence est-elle proche (ou identique) à la séquence biologiquement vraie ? Alors se pose 2 choix possibles : soit nous avons déjà une séquence à laquelle se comparer, soit on détermine la séquence à partir de rien, c'est appelé le séquençage de novo.

3.2.2 Modèles et techniques de contigage

Pour assembler les lectures en contigs, plusieurs méthodes ont été envisagées (voir Annexe B pour plus de détails) :

1. L'assemblage de génome était d'abord vu comme une instance du problème Shortest common superstring (SCS ou plus petite super-string commune). Ce problème consistait, à partir d'un grand nombre de strings, à vouloir trouver la plus petite chaine contenant toutes les lectures au moins une fois. Cette vision du problème était cependant minimaliste et menait à une impasse de par sa complexité (NP-Difficile et Max-SNP Difficile) et parce qu'elle écrase les répétitions,
2. Ensuite, le problème peut être modélisé au travers d'un graphe (voir Figure 7) où les noeuds sont les lectures et il y a un arc entre deux noeuds si leurs chaînes respectives se chevauchent de k caractères. Le problème devient alors une extension du chemin Hamiltonien : trouver un chemin de longueur minimum visitant chaque sommet du graphe une fois. Cette méthode reste très complexe (NP-Difficile) mais utilisé des sccaffoldeurs comme SGA [5].
3. Enfin, la plus utilisée est la modélisation en graphes de De Bruijn. Cette vision repose sur un graphe (voir Figure 7) dans lequel les sommets sont l'ensemble des sous-chaînes de longueur K des lectures, i.e. l'ensemble des K-mer, et il y a un arc entre deux sommets ssi il y a un (k-1)-chevauchement enre les séquences correspondantes. Le problème revient alors toujours à chercher un chemin minimum dans le graphe visitant les noeuds une fois. Cette méthode reste NP-Difficile mais est utilisée par la plupart des assembleurs comme Velvet [6] et Minia [7].

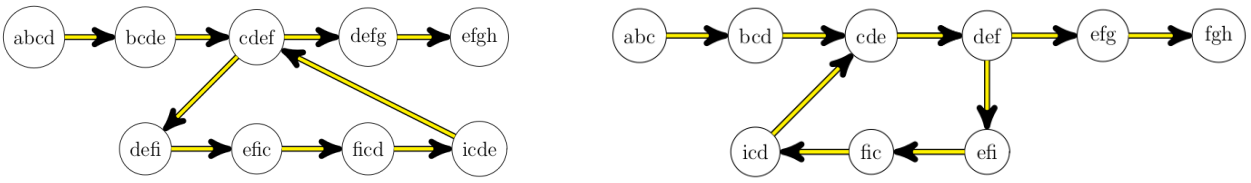

Figure 7: Graphe de string et graphe de De Bruijn avec des lectures de longueur 4 et des K-mer de longueur K = 3.

3.2.3 Modèles et techniques pour le Scaffolding

Modèle général : le graphe de contigs
Le scaffolding est le problème d'**utiliser l'information de connectivité des lectures pairées pour ordonner et orienter les contigs** (plus grandes séquences construites à partir des lectures) dans le génome. Reconstruire le génome en une seule séquence n'est pas toujours faisable à cause des répétitions. Au lieu de ça, un ensemble de contigs est construit à partir des lectures simples et les paires de lectures sont utilisées pour ordonner les contigs. Les paires de lectures nous donnent en effet trois informations :

- l'orientation des lectures sur les contigs nous donne l'orientation relative des contigs 2 à 2,

- le fait que l'on connaisse un sens dans le génome nous permet d'obtenir la précédence entre deux lectures d'une paire et ainsi obtenir une information de précédence entre les 2 contigs que la paire relie : un ordre local,
- le fait que l'on connaisse la distance séparant les deux lectures d'une paires nous donne approximativement la distance entre deux contigs.

On construit une séquence avec les contigs que l'on appelle scaffold. On obtient :

- Le graphe de contig (voir Figure 8 est défini ainsi : V = contigs, E = (c_1, c_2) / $|(r_1, r_2), r_1 \in c_1, r_2 \in c_2| \geq t$ où t est un seuil arbitraire, indiquant que les contigs sont liés si ils sont supportés par au moins t lectures pairées,
- Le problème de scaffolding est alors : trouver une ordre des contigs dans le graphe bi-directionnel des contigs qui est supporté par un nombre maximum de paires de lectures. Le scaffolding nécessite un ensemble complet de contigs,
- Dans la plupart des implantations, l'information de paire est exclusivement utilisée pendant cette étape de scaffolding.

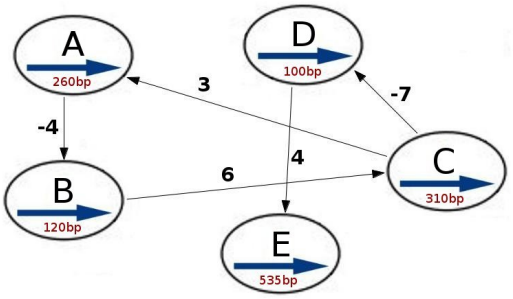

Figure 8: Modélisation d'un problème de scaffolding à 5 contigs. Chaque arête correspond à des paires de lectures liant deux contigs. La pondération sur ces arêtes donne le nombre de paires les reliant avec un signe positif si les contigs ont même orientation, négatif sinon. Le sens de la flèche représente l'information de précédence entre les deux contigs.

En se basant en général sur cette structure, chaque technique utilise ensuite sa propre approche du problème. Examinons 4 des principaux scaffoldeurs existant qui sont ceux sur lesquels j'ai travaillé : Sspace [8], Opera [9], Scarpa [10], Sopra [11] et un dernier qu'à développé le LIRMM avec laquelle nous collaborons : Scaftools [1].

3.3 Scaffolders :

3.3.1 Sspace :

Pour une vue d'ensemble de l'algorithme de Sspace voir Figure 9.

- **Entrée :**
  1. librairies de lecture de tailles d'insert différentes (sui seront traitées de manière hiérarchique en commençant par les librairies de plus petite taille d'insert),
  2. fichiers de contigs.
- **Pré-traitement :**
  1. Les courtes paires de lectures d'ADN sont filtrées en enlevant les séquences contenant des caractères autres que ACTG,
  2. Les lectures restantes sont mappées avec Bowtie [12] sur les contigs pré-assemblés. La position et l'orientation de chaque paire qui a pu être mappée est enregistrée dans une table,
  3. Les paires de lecture dupliquées sont retirées,
  4. (Optionnel) les contigs pré-assemblés sont allongés en utilisant les lectures qui n'ont pas pu être mappées afin d'incorporer les jeux de données non-utilisés dans le mapping.

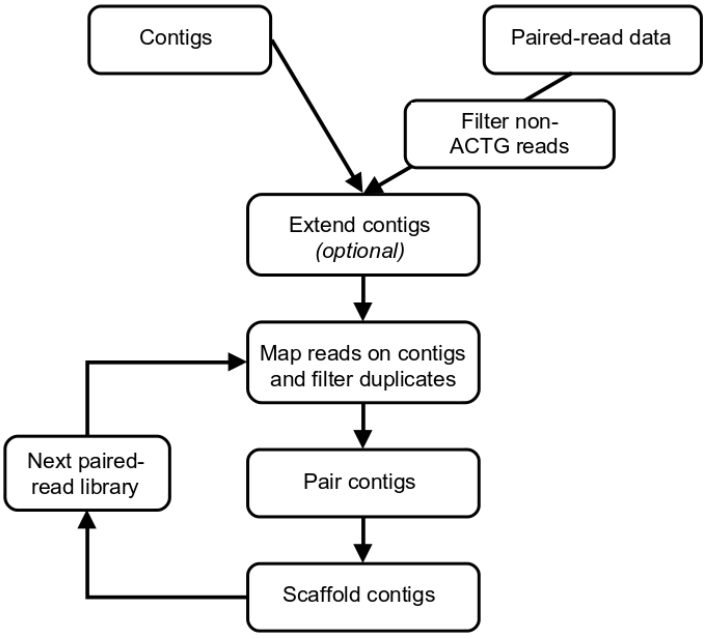

Figure 9: Vue d'ensemble de l'algorithme de Sspace avec ses principales étapes.

- **Scaffolding (extension de SSAKE [13] :**
  1. des paires de contigs supposées (étape de pré-traitement) sont déterminées en se basant sur la position des paires de lectures sur les différents contigs. Les paires de contigs sont seulement considérées si la distance calculée entre eux est comprise dans un intervalle définie par l'utilisateur,
  2. les scaffolds sont formés en combinant itérativement des contigs, en commençant par les plus grands, si il y a un nombre de paires de lectures minimum (k) qui supporte cette connexion (par défaut k=5),
  3. Pour gérer les contigs qui ont des connexions alternatives (Voir Figure 10), si des connexions sont aussi trouvées entre les alternatives elles-même, l'algorithme cherche à les placer dans le bon ordre en se basant sur l'estimation de l'insertion. Ainsi, un quotient est calculé entre les deux meilleures alternatives. Si ce ration est inférieur à un seuil (par défaut a = 0.7), une connexion avec l'alternative de meilleur score est établie,
  4. L'extension d'un scaffold est arrêtée si un contig n'a aucun lien avec d'autres contigs ou si le ratio des alternatives est dépassé et le protocole est répété jusqu'à ce que tous les contigs soient incorporés dans des scaffolds linéaires.
- **Sortie :**
  1. scaffolds finaux (au format FASTA),
  2. un fichier complémentaire qui liste les contigs formant chaque scaffold,
  3. un fichier de résumé contenant des statistiques utiles telles que le nombre total de scaffolds, leur taille (moyenne) et le N50,
  4. (Optionnel) les connections à l'intérieur de chaque scaffold peuvent être visualisées graphiquement.

3.3.2 Opera :

L'approche de ce scaffoldeur se distingue des autre par sa volonté de trouver une solution au problème grâce à une **résolution exacte** avec garantie sur le temps de résolution. En considérant qu'une paire de lecture (respectivement un arc) est concordant(e) dans un scaffold si l'orientation suggérée par elle (lui) est satisfaite et si la distance entre les lectures est inférieure à une taille maximum précisé par la librairie, Opera nous donne cette définition :

Problème de scaffolding : Soit un graphe de contigs G, trouver un scaffold S de contigs qui maximise le nombre d'arcs concordants dans le graphe. Ce problème est prouvé NP-complet.

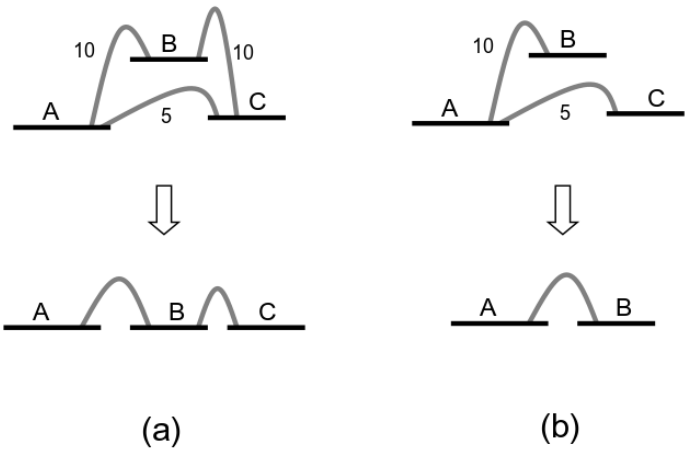

Figure 10: Sspace offre deux scénarios pour le scaffolding de contigs avec connections alternatives. Dans le scénario (a), le contig A a des liens avec les contigs B et C. Puisqu'il existe aussi des liens entre les contigs B et C, et les distances calculées entre tous les contigs sont dans les normes, A, B et C sont placés dans un scaffold. Dans le scénario (b), le contig A a des liens avec les contigs B et C, mais il n'y a pas de liens entre B et C. Ainsi, un ratio est calculé en divisant le nombre de liens trouvé entre A et C (5) par celui trouvé entre A et B (10). Si ce ratio (0.5) est en dessous d'un seuil décidé par l'utilisateur (-a), les contigs A et B sont placés dans un scaffold.

Puis l'approche de l'algorithme est la suivante :

Prétraitement :

1. Suppression des contigs de couverture de lectures trop importante (supérieure à 1.5 fois la moyenne),
2. Construction du graph de contigs avec les contigs comme sommets et des arcs représentants de multiples paires de lecture suggérant une même distance entre contigs et une même orientation relative,
3. Suppression du bruit stochastique introduit par les paires de lecture chimériques en supprimant les arcs dont le nombre de lecture supportant est inférieur à un seuil (ce seuil est déterminé dynamiquement en simulant des paires de lecture chimériques).

Hypothèses et propriétés :
Opera pose comme hypothèse que le problème est borné : comme la distance entre les lectures d'une paire est bornée par τ et que les contigs ont une taille minimum lmin, la largeur de l'arbre de recherche a une borne supérieur w = $\frac{\tau}{l_{min}}$. On s'assure ainsi qu'il est possible de trouver un algorithme de résolution exacte polynomial en la taille du graphe si il n'y a pas d'arc discordant. S'il y en a, le problème de trouver un scaffold avec moins de p arc discordant devient NP-complet mais en posant p comme une constante, on peut espérer trouver un algorithme de résolution exact polynomial en p.

- Soit G(V, E) le graphe de contigs avec V l'ensemble des contigs, E l'ensemble des arcs,
- Soit S' un scaffold partiel : scaffold sur un sous-ensemble des contigs,
- Soit D(S') ensemble en attente de S' : l'ensemble des arcs entre S' et V - S',
- Soit A(S') la région active de S' : suffixe le plus court de S' tel que tous les arcs de D(S') soient adjacent à un contig de A(S'),
- Soit X(S') ensemble discordant : ensemble de tous les arcs discordant dans S'.
- Soit G'(V', E') un graphe clôt de G : les arcs de E entre V - V' et V' sont toujours adjacents à un contig dont la taille est supérieure au seuil τ.

Opera pose ces différentes propriétés :

- Lemma 1 : Considérons deux scaffolds partiels S'$_1$ et S'$_2$ du graphe de contigs G. Si (A(S'$_1$), D(S'$_1$)) = (A(S'$_2$), D(S'$_2$)) alors :
  - (1) S'1 et S'2 contiennent le même ensemble de contigs,
  - (2) les deux ou aucun des deux scaffolds partiels ne peuvent être étendus à une solution.
- Theorem 1 : Soit un graphe de contigs G = (V, E) et un scaffold vide, l'algorithme de scaffolding sans arc discordant a une complexité en temps de O(|E||V|^w).
- Theorem 2 : Soit un graphe de contigs G et la largeur de la librairie w une constante, le problème consistant à décider si il existe un scaffold S avec moins de p arcs discordants est NP-complet.
- Lemma 2 : Soit 2 scaffolds partiels S'$_1$ et S'$_2$ de G avec moins de p arcs discordants. Si (A(S'$_1$), X(S'$_1$)) = (A(S'$_2$), X(S'$_2$)) alors
  - (1) S'$_1$ et S'$_2$ contiennent le même ensemble de contigs,
  - (2) les deux ou aucun des deux scaffolds partiels ne peuvent être étendus à une solution.
- Lemma 3 : Soit un graphe de contigs G = (V, E) et soit p le nombre maximum d'arcs discordants permis. L'algorithme de scaffolding The algorithm Scaffold a une complexité en temps de O(|V|^w |E|$^{p+1}$).
- Lemma 4 : Soit un graphe de contigs G = (V, E), soit G' = (V', E') un graphe clôt de G. Soit S = S'$_1$, ..., S'$_n$ l'ensemble de scaffolds optimal de G' (en déconnectant les scaffolds connectés par des arcs discordants). Il existe un ensemble de scaffolds optimal S de G où chaque S'i est un sous-chemin d'un scaffold de S.

Scaffolding dans Opera :

1. Détermination d'un sous-graphe clôt G' de G minimum,
2. Résolution du problème de scaffolding pour G' pour obtenir les scaffolds {S'$_1$, ..., S'$_n$}: Basé sur les propriétés précédantes, le processus de scaffolding est alors une approche par programmation dynamique basée sur une recherche dans l'espace des scaffolds pour trouver le scaffold optimal. On part d'un scaffold vide puis on l'étend un contig à la fois pour chercher sur la classe d'équivalence des scaffolds partiels, et trouver un scaffold avec le moins d'arcs discordant possible : p. Pour étendre cet algorithme en un algorithme qui va optimiser p, Opera se base sur une approche branch-and-bound où :
   - Une recherche heuristique rapide est utilisée pour trouver une bonne solution et définir une borne supérieure p (pas forcément bonne, on essaye alors les valeurs de p en commençant de 0),
   - La borne est affinée au fur et à mesure que de meilleures solutions sont trouvées,
   - La recherche ne s'arrête que lorsque toutes les extensions ont été explorées pour le scaffold.

   La complexité reste O(|V|^w |E|$^{p+1}$).
3. Remplacer dans G tous les contigs de S'i par un sommet-scaffold pour tout i, on obtient G",
4. Recommencer l'opération avec G" jusqu'à ce qu'on ne puisse pas réduire le graphe,
5. Enfin remplacer dans les scaffolds finaux les sommets-scaffolds par les contigs qu'ils représentent,
6. Pour finir, Opera détermine la taille des espaces entre contigs en résolvant un problème de programmation quadratique.

3.3.3 Scarpa :

- **Entrée :**
  1. fichiers sam contenant le mapping des lectures de différentes librairies sur les contigs (le traitement se fera ensuite sur chaque librairie en commençant par celle de plus petite taille d'insert),
  2. fichiers de contigs.
- **Pré-traitement :**
  1. Filtre le mapping pour ne garder que ceux non-ambigus (un seul mapping),

2. Calcule la moyenne et la déviation de la taille d'insert des librairies de lectures en se servant des contigs plus grands que le N50,
3. Les courtes paires de lectures d'ADN sont filtrées en enlevant les séquences contenant des caractères autres que A, C, T ou G,
4. Les lectures restantes sont mappées avec Bowtie [12] sur les contigs pré-assemblés. La position et l'orientation de chaque paire qui a pu être mappée est enregistrée dans une table,
5. Les paires de lecture dupliquées sont retirées,
6. Construit le graphe de contigs en filtrant les arcs soutenus par moins d'un seuil de lectures (par défaut 2),
7. filtre les sommets en déconnectant les contigs avec un trop grand degré,
8. découpage du graphe en composantes bi-connexes.

- **Scaffolding : orientation**

1. Construction d'un second graphe à partir du graphe de contigs (Voir Figure 11) :
   - chaque sommet c (contig) devient 2 sommets c- et c+ reliés par un arc,
   - chaque arc reliant 2 contigs est lié soit à c+ soit à c- selon que la lecture se trouve mappée à un brin ou l'autre de l'ADN,
   - ces arcs sont également remplassé par 2 noeuds.

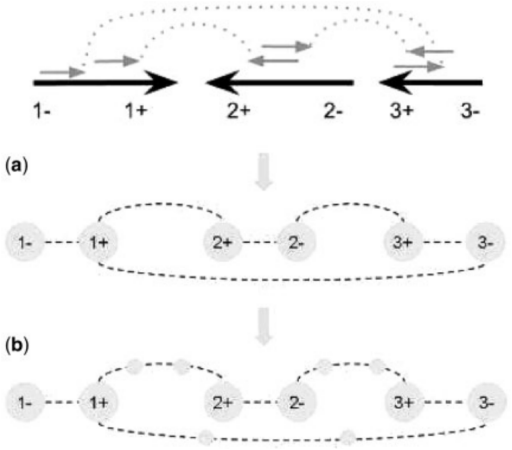

Figure 11: Modélisation du problème d'orientation des contigs comme un problème de calcul de cycle impaire. (a) On crée 2 noeuds pour chaque contig correspondant aux deux extrémités du contig et on connecte ces noeuds par un arc. Puis les liens de paire de lecture sont utilisés pour connecter les extrémités des contigs. Les liens en conflit créent des cycles de longueur impaire dans le graphe obtenu. (b) Pour permettre la suppression de liens de paire de lecture en plus des contigs, on modifie le graphe en créant deux noeuds auxiliaires pour chaque arc induit par ces liens. Cette modification préserve la parité des cycles du graphe original.

2. Le problème de décider l'orientation a alors une solution si et seulement si le graphe n'a aucun cycle de longueur impaire. On recherche donc un minimum de sommets à enlever permettant d'enlever chacun de ces cycles.
3. L'orientation des contigs est enfin décidée en donnant une orientation forward à un contig arbitraire (l'arc entre les deux sommets du contigs est dirigé du x- au x+) et en propageant cette contrainte (Voir Figure 12) :
   - chaque arc sortant d'un sommet x+ doit arriver à un sommet y-,
   - chaque arc sortant d'un sommet x- doit arriver à un sommet y+.

- **Scaffolding : position**

1. Application d'un algorithme à base d'heuristique permettant de supprimer tous les cycles du graphe pour qu'il existe une solution d'ordre,
2. Résolution d'un système en programmation linéaire pour estimer à la fois l'ordre et les espaces entre contigs. Pour chaque lien de paires de lectures et soit x_i et x_j les positions des deux contigs reliés, on introduit les contraintes :

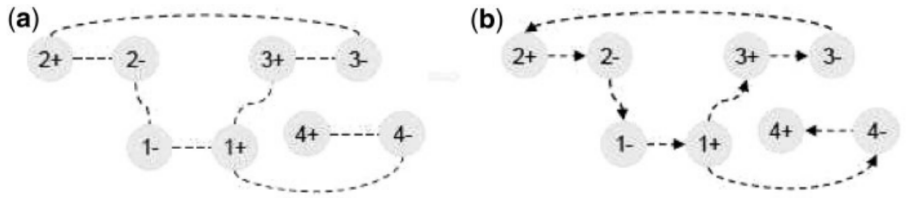

Figure 12: Détermination de l'orientation. (a) Un graphe de contigs non-orienté G après suppression des cycles de longueur impaire. (b) On assine une direction aux arcs d'une manière gloutonne en commençant par un contig arbitraire (dans cet example, du contig 1).

$$x_i - x_j + d_{ij} \leq C(1 - \delta_{ij})$$
$$x_j - x_i - d_{ij} \leq C(1 - \delta_{ij})$$

avec :

- d_{ij} distance entre les extrémités 5' des contigs i et j suggérée par le lien de paire de lectures,
- $\delta_{ij} \in \mathbb{R}$ variables à maximiser,
- C est une grande constante égale à la somme des longueurs de tous les contigs.

3. post-traitement pour vérifier ces espaces (espace trop petit entre contigs non liés, merge de contigs chevauchant en cas de forte identité, etc...)

3.3.4 Sopra :

Pour une vue d'ensemble de l'algorithme de Sopra voir Figure 13.
Considérons que les librairies sont des données Illumina.

- **Entrée :**
  1. librairies de lecture de tailles d'insert différentes (sui seront traitées de manière hiérarchique en commençant par les librairies de plus petite taille d'insert),
  2. fichiers de contigs.
- **Pré-traitement :**
  1. Calcul de la moyenne et la variance des couvertures de lectures des contigs,
  2. Calcul de la taille d'insert empirique des contigs,
  3. Filtre les contigs en supprimant ceux de taille inférieure à un seuil (par défaut 150),
  4. Filtre les contigs de couverture de lectures trop importante (supérieure à la moyenne+h*variance avec h = 2.2 par défaut),
  5. Filtre les paires de lecture en enlevant celles où le nombre d'apparition d'une lecture ou son complément est supérieure à un seuil (par défaut 4),
  6. Filtre les arcs dans le graphe de contigs en enlevant les arcs soutenus par moins d'un certain nombre de paires (par défaut 4),
  7. Idem si la distance estimée entre les 2 contigs est trop grande par rapport à la taille d'insert,
  8. Idem si des paires de lectures indique des informations différentes par rapport au lien entre les deux contigs et que l'on ne peut décider d'une majorité.
- **Scaffolding : orientation**
  1. Découpage du graphe en parties connexes (chacune d'elle donnant un scaffold),
  2. Découpage du graphe en parties biconnexes (parties de graphes séparées par un sommet-articulation),
  3. Découpage des sous-graphes obtenus en clusters :
     - on forme un premier cluster avec un contig arbitraire
     - on construit les autres clusters itérativement à partir des voisins du cluster précédent pris dans les contigs non encore placés.

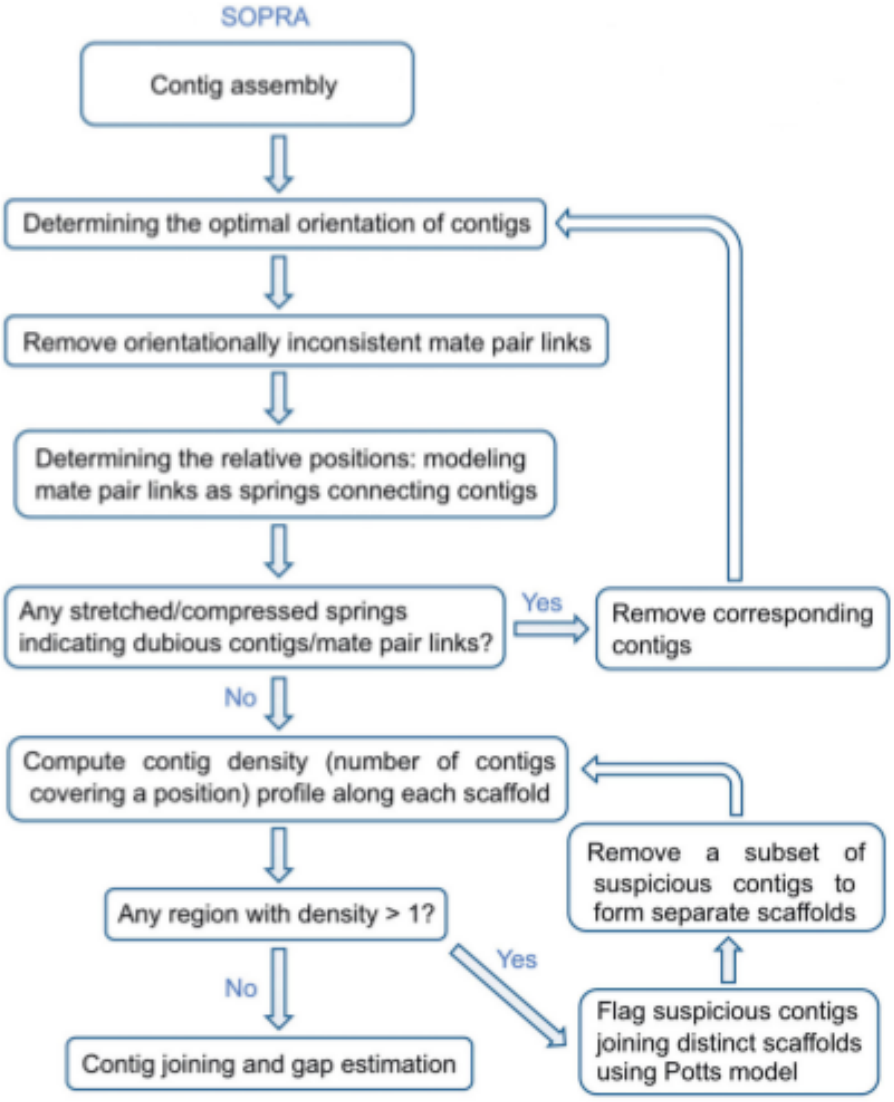

Figure 13: Vue d'ensemble de l'algorithme de Sopra avec ses principales étapes.

4. Résolution de l'orientation dans les sous-graphe :
   - par programmation dynamique si le nombre maximum de contigs dans un cluster du sous-graphe est inférieure à un seuil (par défaut 6),
   - par recuit simulé sinon.
5. Les sous-graphes sont remis ensemble pour retrouver l'orientation dans le scaffold.

- **Scaffolding : position**

1. Le système est modélisé en problème de ressorts à 1 dimension pour chaque scaffold (voir Figure 14), où chaque contig est représenté par une position (ou masse) et chaque arc est un ressort. On obtient un système linéaire de la forme AX = b où A est la matrice d'adjacence du graphe et les coefficients de b sont la somme des longueurs avec les autres contigs liés pour chaque contig,
2. La résolution du système se fait au travers d'un gradient conjugué avec un nombre d'itérations égal à Min(6000, nombre_de_contigs),
3. On obtient alors la position de chaque contig dans le scaffold.
4. On vérifie si chaque longueur entre 2 contigs était celle attendue :
   - si ce n'est pas le cas on enlève le contig de plus petite taille dans chaque paire de contig à problème et le processus reprend au découpage du graphe en parties biconnexes,
   - sinon les scaffolds sont prêts.
5. Pour finir, si des scaffolds ont une densité > 1 (il y a des région où plusieurs contigs se chevauchent), on va découper ces régions afin de tenter d'obtenir des scaffolds plus linéaires,

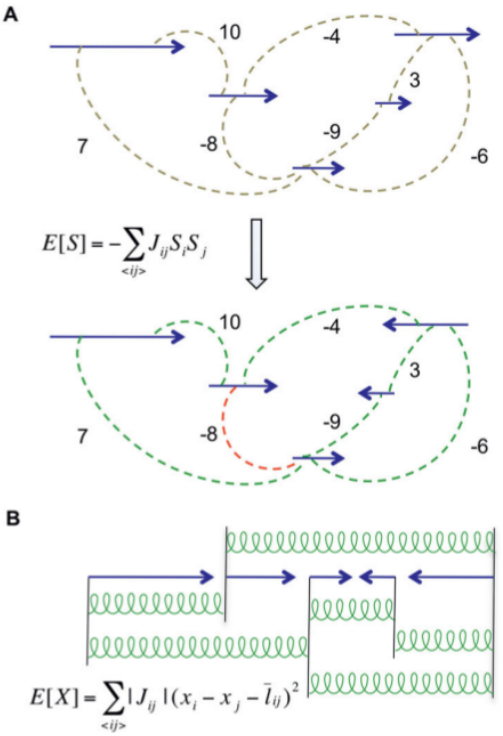

Figure 14: Modelisation des contraintes sur les arcs dans le graphe de contigs. (A) Pour deux contigs i et j connectés via des paires de lecture, la quantité J_{ij} représente l'information sur l'orientation relative (signe de J_{ij}) et le nombre de paires de lecture connectant les 2 contigs (valeur absolue de J_{ij}). Minimiser l'énergie produit une assignation de l'orientation qui satisfait le plus de contraintes possible. Les contraintes qui ne sont pas satisfaites dans la configuration optimale (en rouge) sont ignorées dans la suite. (B) Pour déterminer la position relative des contigs, on modélise l'ensemble des lectures connectant i et j comme un ressort attaché aux points de départ de ces contigs. La longueur à vide de ce ressort, l_ij, est égale à la valeur moyenne des distances suggérées par les paires de lecture.

6. post-traitement pour vérifier ces espaces (espace trop petit entre contigs non liés, merge de contigs chevauchant en cas de forte identité, etc...)

3.3.5 Scaftools :

Modèle :
Soit un ensemble de contigs C = {C_0, ..., C_{n-1}} et un ensemble de paires de lecture, on veut décider de l'ordre et de l'orientation des contigs qui respectent le plus de lectures possible. En supposant que les lectures ont bien été mappées aux contigs et que l'on se concentre sur des paires mappées sur des contigs distincts. Il y a 4 *configurations* possibles d'orientations pour les contigs.

Plusieurs paires peuvent soutenir la même *configuration*. Dans ce cas, on suppose que l'on peut les grouper et attribuer un poid à cette *configuration* égal au nombre de paire de ce type. On obtient un graphe de contigs pondéré. Comme les contigs sont des séquences orientées, ils sont représentés par une paire de sommets liés par un arc intra-contig. Le contig numéro k est donc représenté par 2 sommets que l'on numérote 2k et 2k+1 (voir Figure 16 pour un exemple).
Le génome peut être composé de plusieurs chromosomes, linéaires ou circulaires. Un modèle général doit préférablement être adopté qui permettra de gérer ces différents cas. Dans ce qui suit, le problème est modélisé tel qu'une solution produise un sous-graphe du graphe de contigs qui n'a que des parties connexes linéiares ou circulaires (chemins ou cycles), et maximise le poid de la *configuration* choisie (i.e. les arcs pondérés du graphe de contigs).

Le LIRMM a alors choisi une approche en programmation linéaire en nombres entiers (ILP pour Integer Linear Programming) pour résoudre le problème. Comme les poids des arcs inter-contigs sont des entiers, ce modèle semble plutôt naturel. L'ensemble des variables est juste un ensemble de *configurations*, notées V = {x_1,

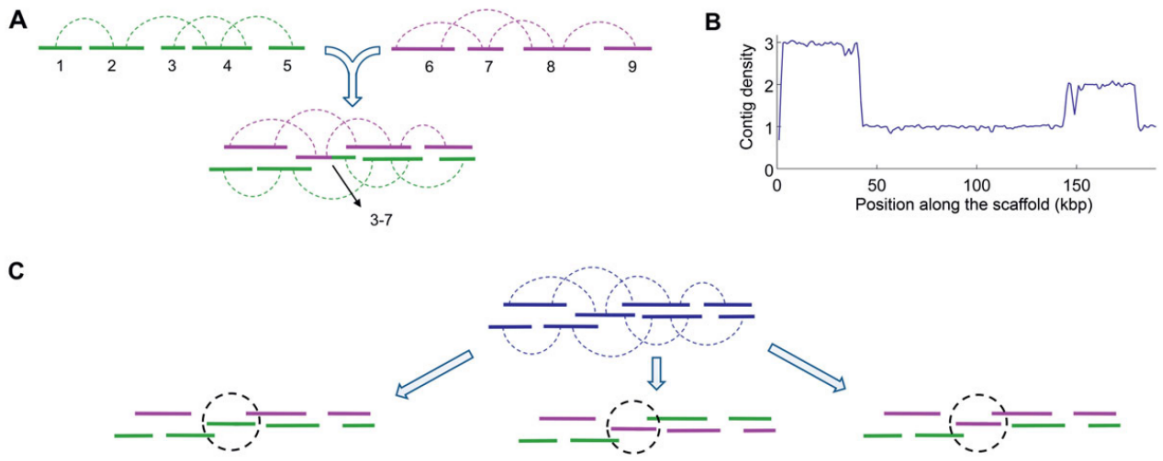

Figure 15: Détection et détermination des erreurs d'assemblage dans les scaffolds en utilisant leur profile de densité. (A) 2 scaffolds, en vert et violet, appartiennent à différentes régions du génome. L'erreur d'assemblage d'un contig chimérique composé du contig 3 du scaffold vert et du contig 7 du scaffold violet a provoqué la fusion des deux scaffold. Dans le nouveau scaffold, plusieurs positions sont couvertes par 2 contigs. (B) Pour un scaffold réel, le profile de densité devrait être proche de 1 (ou 0 pour des gaps). Le tracé montre le profile de densité pour un scaffold mal assemblé obtenu dans le processus d'assemblage d'un jeu de données réel pour le génome E. coli. Chaque point le long de l'axe x représente les positions le long d'une portion de 1000bp du scaffold. L'axe y montre la densité relative pour les positions au long de cette portion. A partie de ce profile, on peut trouver qu'au moins 4 scaffolds ont été assemblés ensemble par erreur. (C) La méthode de marquage pour diviser les contigs en groupes distincts pour le cas montré en (A) peut mener à n'importe laquelle des 3 possibilités montrées ici. On utilise des couleurs pour montrer différents marquage. notons que le contig problématique (3-7) repose toujours à la bordure entre les différents groupes.

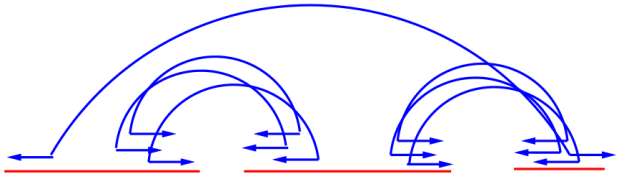

Figure 16: Exemple de graphe de contigs a 6 sommets (k=3) et trois arcs inter-contigs (0, 5), (1, 2), (3, 4) avec des poids 1, 3, 3 respectivement (m=3) et trois arcs intra-contigs (0, 1), (2, 3), (4, 5).

. . . , x_m}. Le domaine de ces variables est D = {0, 1}, où 0 signifie que la *configuration* a été choisie, et 1 sinon.

L'ensemble des contraintes C exprime la contrainte sur le degré du sous-graphe choisi, i.e. un ensemble de chemins et de cycles. Pour chaque extrémité de contig, on impose qu'au maximum une *configuration* sortante soit choisie. Nous ne posons pas comme contrainte que la solution soit connexe il faut éviter de choisir un chemin artificiel incluant des *configurations* de poids trop petit. Finalement, on maximise la fonction objectif correspondante au poid du sous-graphe choisi : $\sum_{j=1}^{m}(w_j x_j)$ où w_j est le poids de la *configuration* x_j.

Dans la Figure 17, on présente un graphe de contigs avec n = 7 contigs (14 sommets). Le modèle ILP suit l'ensemble de contraintes suivant :

$$\begin{array}{lcl@{\qquad}lcl} x_1 & \leq & 1 & x_8 & \leq & 1 \\ x_2 & \leq & 1 & x_8 + x_9 & \leq & 1 \\ x_3 & \leq & 1 & x_4 + x_7 & \leq & 1 \\ x_3 + x_4 + x_5 & \leq & 1 & x_9 + x_{10} & \leq & 1 \\ x_1 + x_6 & \leq & 1 & x_{10} & \leq & 1 \\ x_6 + x_7 & \leq & 1 & x_2 + x_5 & \leq & 1 \end{array} \tag{7}$$

Et la fonction objectif suivante à maximiser :

$$32x_1 + 2x_2 + 2x_3 + 28x_4 + 52x_5 + 20x_6 + 80x_7 + 6x_8 + 34x_9 + 4x_{10} \tag{8}$$

Une solution optimale à ce problème a une valeur objectif de 198, en sélectionnant les arcs x_1, x_5, x_7 et x_9 assignés à 1. Le scaffolding résultant contient 3 scaffolds avec l'ordre (C_1), (C_6, C_2, C_0), and (C_5, C_4, C_3).

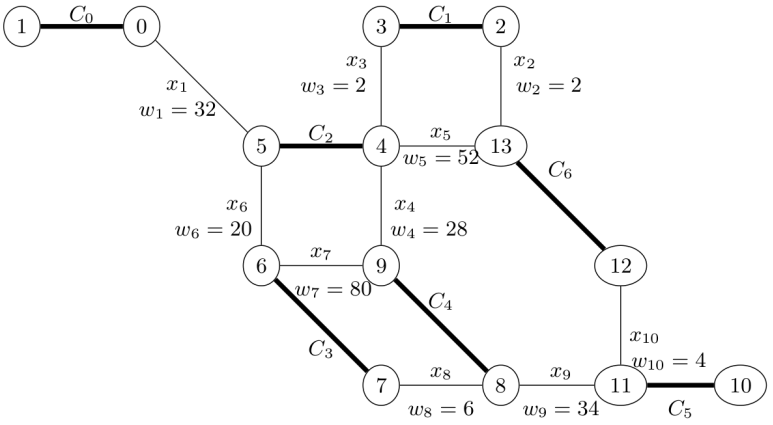

Figure 17: Exemple de graphe de contigs avec n = 7

Ce modèle peut être résolu en utilisant plusieurs types de solvers (SAT, Max-SAT, ILP ...) et l'équipe a choisi le solver de l'état de l'art en branch-and-bound IBM ILOG cplex. Ce modèle s'est en effet prouvé très efficace.

Cycles removing :
Dans notre modèle, seules des contraintes sur le degré de chaque sommet sont utilisées, des solutions comportant des cycles sont possibles. Excpeté un cycle Hamiltonien, qui pourrait correspondre à un génome circulaire, ces cycles doivent être supprimés. Nous supposons aussi que ces cycles sont générés par des répétitions dans le génome. Si deux contigs a et b forment une tandem repeat dans le génome (ils apparaissent plusieurs fois dans le génome consécutivement dans cet ordre, par exemple ababab) alors les *configurations* "a est suivi de b" et "b est suivi de a" ont des poids exagérés comparé à des *configurations* qui ne sont pas dupliquées. Ainsi, le cycle aba pourrait être choise au lieu de la séquence ab incluse dans une séquence plus longue. Notre phase de suppression de cycle est alors aussi une potentielle phase de suppression de répétitions.

Plusieurs encodage ont été proposés pour forcer la contrainte "pas de sous-cycle" en CSP, mais une telle contrainte est moins facile à exprimer dans un contexte ILP. L'équipe a donc décidé de s'attaquer à ce problème avec une technique inspirée de la SAT Modulo Theory : on résout le problème ILP avec un solver puis on intègre cette résolution dans un processus incrémental, où les cycles qui ont été produits par le solver sont interdits dans l'itération suivante.

3.4 Problématique de recherche

3.4.1 Problématique

Il y a plusieurs problèmes inhérents au scaffolding mais le principal vient de la nature de la donnée de séquençage et du génome à la base :

1. **Les répétitions :** Le génome est composé de seulement de 4 nucléotides qui s'enchaînent et les possibilités limitées de permutations entre ces 4 caractères induit des répétitions dans le génome. Ces répétitions vont être difficilement détectables et même impossibles à détecter, au moment du contigage tout du moins, si leur taille dépasse celle des lectures.
2. **Incertitude sur la donnée :** La donnée en entrée (les lectures) provient de processus microbiologiques complexes et impossibles à maîtriser à 100%, surtout avec les contraintes financières, ainsi il y a et y

aura toujours des erreurs dans les données. Ces erreurs peuvent être des lectures erronées (remplacement de nucléotide, ...) ou encore des paires chimériques (2 séquences indiquées comme séparées par un espace X alors qu'il n'en est rien). Cependant, l'incertitude la plus grande repose sur la taille des espaces entre les 2 séquences d'une paire : cet espace n'est connu qu'à une certaine incertitude près et va ainsi varier d'une paire à l'autre suivant une loi normale.

3. **Erreur accumulée :** Bien avant le scaffolding, les erreurs précédentes vont conduire à des erreurs de contigage. On va notamment créer des contigs chimériques, c'est à dire des contigs qui n'ont aucune existence dans la réalité biologique. Cette sorte de contigs est créé notamment à cause des répétitions qui vont conduire à mettre bout à bout des lectures "étrangères". On voit ainsi que les erreurs du départ conduisent à des erreurs plus grosses au moment du contigage et ces erreurs risquent d'être encore plus accentuées après scaffolding.

La problématique est alors : comment pourrait-on modéliser et résoudre le problème de scaffolding de manière à d'une part accélérer les calculs et d'autre part prendre en compte l'incertitude sur la donnée sans pour autant perdre d'information concernant le génome de départ ? On appréhende déjà ici le fait qu'une solution de scaffolding qui respecterait toutes les données de paire n'existe pas : on va essayer de trouver la "meilleure" solution. Il faudra alors également caractériser la notion de "meilleure" solution mais également pouvoir la comparer avec d'autres solutions.

3.4.2 Etat de l'art

Assez naturellement, on comprend que déterminer l'ordre et l'orientation des contigs est un problème combinatoire comme l'on cherche la meilleure solution parmi l'ensemble des permutations sur l'ordre et l'orientation des contigs. Ce problème a été prouvé NP-difficile et on ne connaît donc pas d'algorithme avec une complexité polynomiale qui permettrait de le résoudre.

Pour ce qui est de calculer la taille des espaces entre les contigs, on utilisera des modèles probabilistes (la répartition des tailles en loi normale va notamment rentrer en jeu) et on ramène le tout à un problème d'optimisation linéaire ou quadratique.

Il existe déjà des techniques de scaffolding qui adoptent toutes des approches très différentes. Pour résumer les techniques que j'ai étudiées :

Scaffolder	Phase	Programmation	Domaine
Sopra [11]	Orientation	Dynamique	Discret
	Position	Linéaire	Continu
Opera [9]	Orientation & Ordre	Dynamique	Discret
	Espacement	Quadratique	Continu
MIP [14]	Toutes	Linéaire mixte	Discret et Continu

Ces scaffoldeurs existent et il y a besoin de techniques pour les évaluer et les comparer, sinon comment tirer les avantages de chacun ? Des papiers donnent des pistes concernant cette évaluation (voir notamment [15], [16], [17] et [18]). La technique dont j'ai le plus étudier le fonctionnement interne et que j'ai adapté au cadre des WCSP (dans une première approche) est Sopra, [11].

4 Ma contribution

Je vais présenter dans cette section toute ma contribution pendant ce stage et l'état d'avancement par rapport aux objectifs.

4.1 Evaluation de scaffoldeurs

Ma première tâche en tant que stagiaire dans l'unité aura été de découvrir le problème et d'effectuer un gros travail bibliographique dans les domaines de la génomique et de l'optimisation combinatoire. Les résultats de ce travail ont été donnés dans les 2 sections précédentes.

Durant tout ce travail bibliographique, et aux travers des premières expérimentations effectuées que je présenterai dans la prochaine section, un thème est ressorti : l'évaluation des scaffoldeurs. Comment savoir si tel ou tel scaffoldeur est meilleur qu'un autre pour une application particulière ? Est-ce que la longueur des scaffolds suffit ? Certainement pas puisqu'à l'intérieur des scaffolds la longueur des espacements, l'ordre et l'orientation peuvent être complètement erronés par rapport à la réalité biologique (en raison d'une inertitude sur les données) et on a donc besoin de métriques particulières pour évaluer ces scaffolds.

Ainsi de nombreux papiers ont été écrits sur ce thème de l'évaluation et chaque article sur un scaffoldeur présente une comparaison avec les assembleurs existants à sa manière. Une de mes contributions a alors consisté en une réflexion sur l'ensemble des métriques disponibles afin de pouvoir en sortir les plus pertinentes (voir [16], [18], [19], [20], [21], [15], [16] et les articles des scaffoldeurs pour toutes ces métriques).

4.1.1 Approches d'évaluation

Evaluer des scaffolds produits par différents scaffoldeurs consiste en fait généralement à comparer ces scaffolds. On va alors essayer un maximum d'avoir des conditions identiques pour chaque scaffoldeurs (même contigs, même analyse), à commencer par les mêmes librairie :

- **numéro d'accession**,
- **nombre de lectures**,
- **nombre de paires de lectures**,
- **taille des lectures**,
- **taille d'insert**,
- **couverture moyenne** : nombre moyen de lectures couvrant chaque endroit du génome.

Cependant, on ne peut jamais vraiment comparer 2 scaffoldeurs sur un pieds d'égalité car certains seront plus sensibles aux différences de bibliothèques ou encore aux paramètres utilisés qui ne peuvent donc être identiques.

Il y 3 situations pour l'évaluation :

- soit on n'a **aucun à priori** sur le génome,
- soit on connait un **génome de référence** pour l'espèce séquencée,
- soit on possède d'**autres informations** sur le génome.

Chacun de ces cas impose ses propres métriques et approches que je vais présenter.

4.1.2 Génome inconnu

Sans à priori sur le génome, nous pouvons tout de même obtenir des informations sur la qualité de l'assemblage et la performance du scaffoldeur. Dans la suite, les métriques concernant les scaffolds sont applicables aux contigs (lorsqu'on veut les évaluer). En voici quelques unes des plus utilisées :

- Qualité de l'assemblage :
    1. **le nombre de scaffolds**,
    2. **la longueur du plus long scaffolds** (bp),
    3. **la longueur cumulée des scaffolds** (bp),

4. **la quantité GC dans les scaffolds** (%),
5. **Nx** (bp) avec $0 \leq x \leq 100$ (bp) : longueur maximum telle que la longueur cumulée des scaffolds de longueur supérieure ou égale produit au moins x% de la longueur cumulée des scaffolds,
6. **Lx** ($0 \leq x \leq 100$) : nombre minimum de scaffolds qui produit x% de la longueur cumulée des scaffolds, correspond au nombre de scaffolds de longueur au moins Nx.
7. le coefficient $\mathbf{E} = \sum cLc^2/Lg$ avec Lg la longueur estimée du génome (somme des longueurs des contigs). Ce coefficient répond à la question : combien de gènes en un morceau seront complètement contenus dans les scaffolds ?
8. **nombre de N** dans les scaffolds,
9. **nombre de N par 100kbp** dans les scaffolds.

- Performance :
  1. **Temps d'exécution** (s),
  2. **Mémoire virtuelle** maximum nécessaire (GB),
  3. **Caractéristiques de la machine** sur laquelle on a exécuté l'assemblage.

L'article REAPR [17] tente une évaluation des assemblages sans connaissance du génome à priori en passant par une étude de la couverture sur l'assemblage, voir Figure 18 pour un aperçu de cette méthode. Cela a donné naissance à un coefficient : $Numberoferrorfreebasesx\frac{(brokenN50)^2}{originalN50}$ qui permet de récompenser la précision locale, la contiguité globale et la correction du scaffolding dans un assemblage.

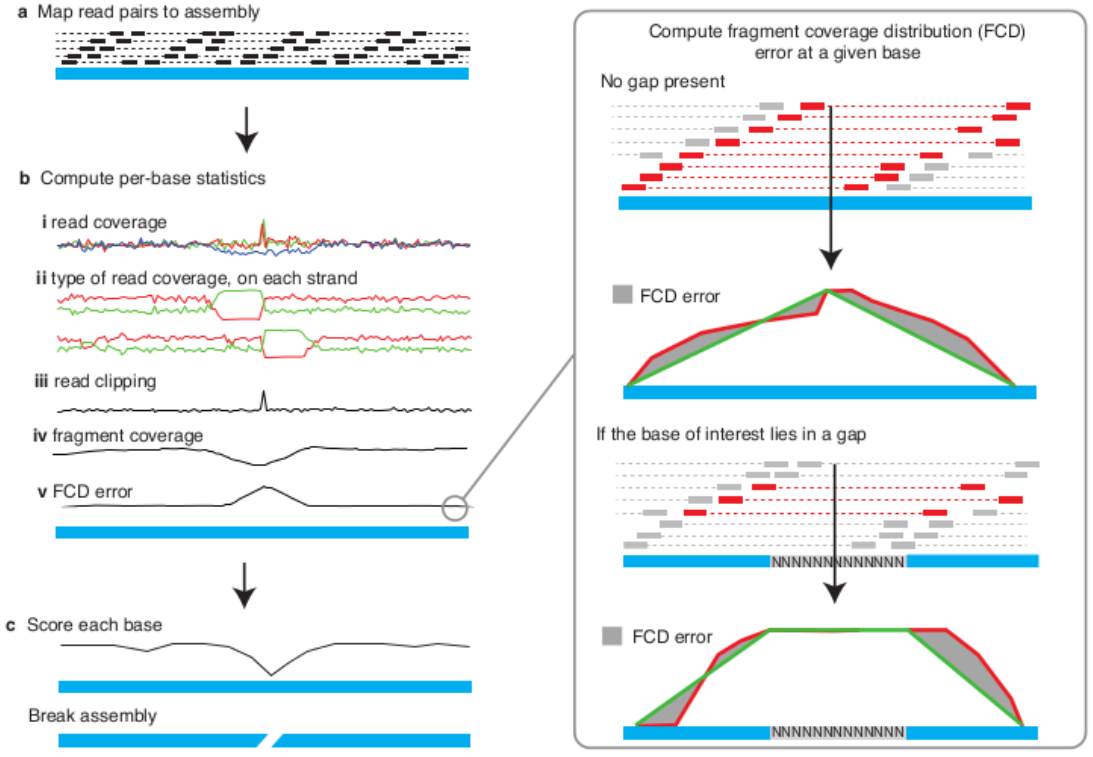

Figure 18: Vue d'ensemble du pipeline de REAPR. (a) L'entrée est un fichier BAN de paires de lecture mappées sur l'assemblage. (b) Les statistiques sont calculées à chaque base du génome : (i) la couverture en lecture par brin, et tout mapping de lecture parfait et unique est incorporé ; (ii) Le type de couverture en lecture sur le brin forward (tracé du haut) et reverse (tracé du bas) : proportion de lectures qui sont réellement pairées (rouge), ou orphelines (vertes) ; (iv) la couverture du fragment, déterminée par les paires réelles : (v) l'erreur FCD, prenant en compte la présence de gap. Dans l'encadré : le calcul du FCD à une base donnée. Les fragments couvrant la base, montrés en rouge, sont utilisés pour construire un tracé de profondeur de fragment (rouge). L'erreur FCD est la zone (grise) entre le tracé observé (rouge) et le tracé idéal (vert). Comme aucune lecture ne se mappe à un gap dans l'assemblage, le calcul est corrigé quand un gap est présent. (c) Les statistiques à chaque base sont utilisées indépendamment pour assigner un score à chaque base de l'assemblage et également casser l'assemblage aux erreurs de scaffolding.

4.1.3 Génome connu

Lorsque le génome de référence est connu de nouvelles métriques peuvent être calculées qui nous donnent plus d'information sur la qualité de l'assemblage :

- **Statistiques sur le génome** :
  1. **Longueur du génome** (bp),
  2. **# chromosomes dans le genome**,
  3. **la quantité GC dans le génome** (%),
  4. **NGx** (bp) avec $0 \leq x \leq 100$ (bp) : longueur maximum telle que la longueur cumulée des scaffolds de longueur supérieure ou égale produit au moins x% de la longueur du génome de référence,
  5. **LGx** ($0 \leq x \leq 100$) : nombre minimum de scaffolds qui produit x% de la longueur du génome, correspond au nombre de scaffolds de longueur au moins Nx,
  6. **K-mer uniqueness** (%) : percentage of the genome covered by unique sequences of length K,
  7. **Genome fraction** (%) : Nombre total de paires de base alignées dans la référence, divisée par la taille du génome. Une base est considérée comme alignée si il y a au moins un scaffold avec au moins un alignement sur cette base. Les scaffolds provenant de régions répétées peuvent apparaître plusieurs fois dans ce calcul,
  8. **Duplication ratio** : Nombre total de bases alignées dans l'assemblage (i.e. Total length - Fully unaligned length - Partially unaligned length), divisé par la taille du génome. Si l'assemblage contient beaucoup de scaffolds couvrant la même région du génome, son Duplication ratio peut être plus grand que 1. Cela peut arriver en raison d'une surestimation de la multiplicité des répétitions et à de petits chevauchements entre scaffolds, parmi d'autres raisons,
  9. **Plus long alignement** (bp) : longueur du plus long alignement continu dans l'assemblage. Cette métrique peut être plus petite que le plus grand scaffold si il contient des erreurs d'assemblage,
- **Erreurs d'assemblage** :
  1. **# scaffolds contenant des erreurs d'assemblage** (sauf scaffolds partiellement non-alignés),
  2. **# erreurs d'assemblage** : nombre de positions dans les scaffolds où :
     - 2 séquences collées sont séparées dans la référence (relocation),
     - 2 séquences séparés dans la référence se chevauchent sur plus de 1kbp (relocation),
     - 2 séquences collées sont sur différents brins dans la référence (inversion),
     - 2 séquences collées sont sur différents chromosomes dans la référence (translocation).
  3. **# relocations**,
  4. **# translocations**,
  5. **# inversions**,
  6. **Longueur cumulée des scaffolds avec erreur d'assemblage**,
  7. **# Erreurs d'assemblage locales** : nombre d'erreurs d'assemblages locale. Une erreur d'assemblage locale est définie comme un endroit où :
     - 1. 1 alignements ou plus couvre cet endroit,
     - 2. l'espace entre les 2 séquences de l'erreur est de moins de 1kbp,
     - 3. les 2 séquences sont sur le même brin du même chromosome du génome de référence.
  8. **NAx** (ou **Nx corrigé**) : Nx où la longueur des blocs alignés sont utilisés au lieu de la longueur des scaffolds, i.e. si un scaffold a une erreur d'assemblage, le scaffold est coupé à cet endroit,
  9. **NGAx**, **LAx**, **LGAx** (ou **NGx**, **Lx**, **LGx corrigés**): idem avec les NGx, Lx, LGx.
- **Unaligned** :
  1. **# scaffolds totalement non-alignés** : nombre de scaffolds qui n'ont aucun alignement sur la référence,
  2. **longueur des scaffolds totalement non-alignés** : longueur cumulée des scaffolds totalement non-alignés,
  3. **# scaffolds partiellement non-alignés** : nombre de scaffolds dont certains fragments n'ont aucun alignement avec la référence,

- **# avec erreur d'assemblage** : nombre de scaffolds partiellement non-alignés qui contiennent des erreurs d'assemblages dans des fragments alignés. Ces erreurs d'assemblages ne sont pas comptées dans la métrique # erreurs d'assemblages.
- **# both parts are significant** : nombre de scaffolds partiellement non-alignés qui contiennent à la fois des fragments alignés et non-alignés de longueur $\geq$ 500bp.

4. **longueur des scaffolds partiellement non-alignés** : longueur cumulée des scaffolds partiellement non-alignés.

- **SNPs** :

1. **# SNPs** : nombre de SNPs dans toutes les bases alignées,
2. **# SNPs pour 100kbp** : nombre moyen de SNPs par 100kbp,
3. **# indels** : nombre d'indels (insertion/deletion) dans toutes les bases alignées,
4. **longueur cumulée des indels**,
5. **# indels pour 100kbp** : nombre moyen de indels par 100kbp,
   - **# courts indels** : nombre d'indels de taille inférieure ou égale à 5bp,
   - **# longs indels** : nombre d'indels de taille supérieure à 5bp.
6. **# N's** : nombre total de base indécidées N (dans des espaces entre contigs le plus souvent) dans l'assemblage,
7. **# N's per 100 kbp** : nombre moyen de N par 100kbp.

Voir Figure 19 pour différentes représentations possibles grâce à ces métriques.

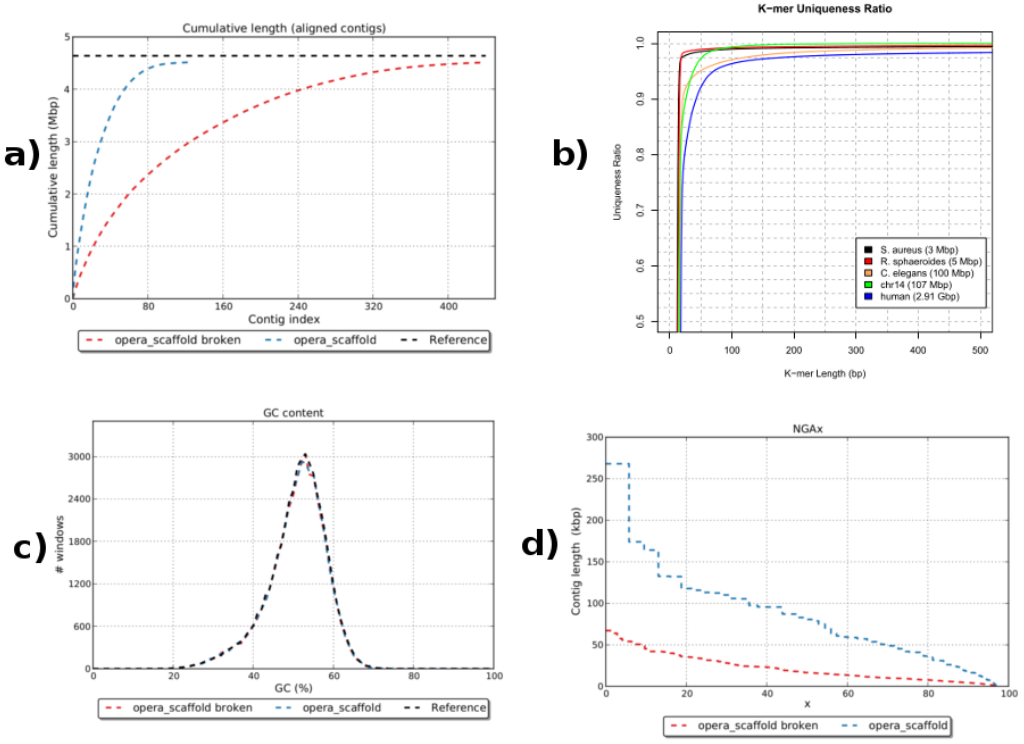

Figure 19: Représentation possibles grâce à ces métriques. (a) Cumulative plot. (b) K-mer uniqueness ratio : pourcentage du génome couvert par des séquences uniques de longueur K. (c) GC content : pourcentage de GC moyen en fonction de la taille de séquence prise. (d) NGA plot : tracé de la taille de scaffold en fonction du pourcentage x pour NGAx.

L'article d'évaluation [18] propose une méthode original permettant d'évaluer des assemblages en plus de certaines des métriques précédentes :

- Construire des exemples tests de graphes de contigs - ou cas d'utilisation - dont on souhaite une des solutions, pour observer le comportement de chaque scaffoldeurs (voir Figure **??**),

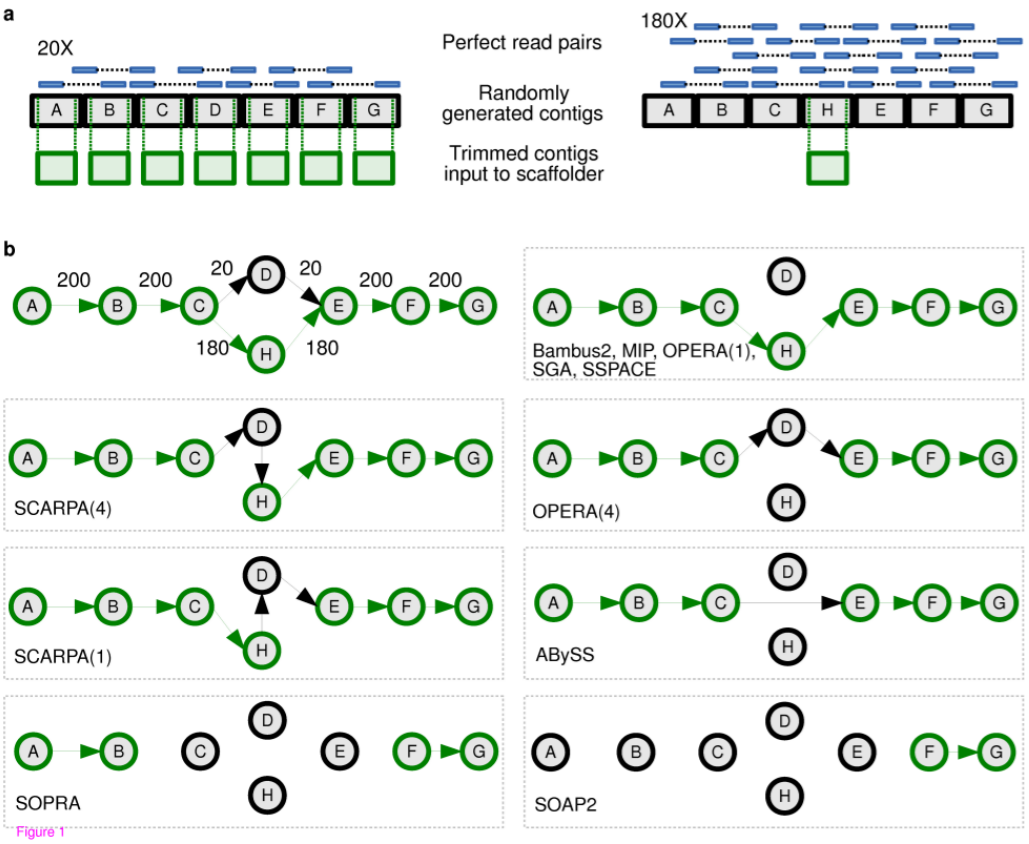

Figure 20: Jeu de tests de Hunt avec les résultats de chaque scaffolder testé.

- Dans chaque contig, on trouve une séquence assez longue pour être unique dans le génome de référence. Cette séquence, ce tag, va servir à repérer le contig dans les scaffolds assemblés et on peut ainsi retrouver l'ordre et l'orientation des contigs avec exactitude dans le scaffold. De cette manière on vérifie le comportement du scaffoldeur sur les exemple précédents,
- Pour finir, l'article définit 4 métriques supplémentaires pour évaluer la qualité de l'assemblage par rapport aux tags :
  - Correct joins : nombre de contigs joints à raison,
  - Incorrect joins : nombre de jointures erronées,
  - Skipped tags : nombre de contigs joints (i.e. séparés par un espace) alors qu'il aurait du y avoir un autre contig entre eux,
  - Lost tags : nombre de contig (ou bout de contig contenant le tag) perdus.

4.1.4 Autres sources d'information

En plus du génome de référence (ou à la place), d'autres sources dinformations peuvent être acquises pour contrôler la qualité de scaffolds. Prenons ceux présentés dans [16] :

- **Les fosmids** qui nous donnent des séquences sûres que l'on peut comparer avec notre assemblage (voir Figure [FIG]). Soit :
  - $(S_i)_{i\in\{1,..,n\}}$ les longueurs des scaffolds,
  - $(R_i)_{i\in\{1,..,n\}}$ les longueurs des séquences de fosmids,
  - $(A_i)_{i\in\{1,..,n\}}$ les longueurs des alignements des scaffolds sur les séquences de fosmids,
  - $(C_i)_{i\in\{1,..,n\}}$ les longueurs des "îlots couverts" (zone d'une séquence de fosmids complètement couverte par un ou plusieurs alignements de scaffolds.

  A partir de ces coefficients, on calcule 4 métriques :

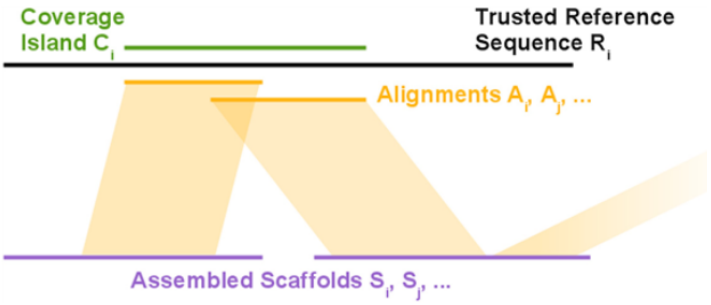

Figure 21: Alignement des scaffolds sur les fosmids.

  - **Couverture** = $\frac{\sum C_i}{\sum Ri}$: quelle fraction des séquences de fosmids ont été assemblées ?
  - **Validité** = $\frac{\sum A_i}{\sum Si}$: quelle fraction de l'assemblage peut être validée par les séquences de fosmids,
  - **Multiplicité** = $\frac{\sum A_i}{\sum Ci}$: est-ce que les répétitions ont été répliquées ou écrasées durant l'assemblage ?
  - **Parsimonie** = $\frac{\sum S_i}{\sum Ci}$: quel est le coût de l'assemblage (bp assemblées / bp validées) ?
- **Les cartes optiques**,
- **La location de gènes dans l'assemblage** : il existe des bibliothèques de gènes retrouvés très fréquemment que ce soit chez les eukaryotes ou les prokaryotes. Ces gènes on est quasiment sûre d'en trouver dans le génome séquencé, on va donc regarder si dans l'assemblage ils sont présents. On obtient 2 métriques très utilisées :
  - **# CGs** : nombre de "Core Genes" (genes fréquemment rencontrés) trouvés dans l'assemblage,
  - **# protéines** : nombre de protéines pouvant être synthétisées à partir des gènes dans l'assemblage.

4.1.5 Conclusions

Après l'étude des différents articles sur le sujet, il m'a fallut faire le tri des différentes métriques à utiliser pour comparer différents scaffoldeurs dans mes expérimentations. On m'a notamment demandé d'apporter les résultats expérimentaux dans le cadre d'un article écrit avec une équipe du LIRMM. Voici donc ce que je préconise d'afficher dans le cas où on connait le génome de référence :

- **Performance** : Caractéristiques de la machine, Temps d'exécution et Mémoire virtuelle maximum nécessaire (si disponible).
- **Jeux de données** : numéro d'accession, # paires de lectures, taille des lectures et taille d'insert.
- **Génome** : longueur, # chromosomes, % GC, Genome fraction.
- **Scaffolds** : # scaffolds, longueur cumulée des scaffolds, NG50, NG50 corrigé, # erreurs d'assemblage, (Optionnel : # relocations, # translocations, # inversions), # Local misassemblies et # N's.
- **Contigs** : idem sauf le comptage des N's (il est important de garder une trace des données de départ pour comparer).

C'est ces métriques que nous avons mis dans l'article pour comparer avec d'autres scaffoldeurs. Elles permettent selon d'attester à la fois de l'efficacité du scaffoldeurs (plus l'assemblage est long et le NG50 grand plus le scaffoldeur est efficace) et sa précision (via les erreurs d'assemblages). Il est également très important de permettre au lecteur de pouvoir reproduire ces résultats en donnant les informations sur le jeu de données etc... Et surtout, les informations sur la machine permettent de rendre crédibles les résultats. Plus d'un papier s'en passe et présente des résultats en temps incroyables. Pour classer les scaffoldeurs, il faudra alors décider des meilleurs dans chaque métrique et mettre une pondération à chacune afin d'exprimer si l'on désire plutôt pénaliser des assemblage aggressif (grand NG50 mais beaucoup d'erreurs) ou le contraire.

En plus de ces métriques, toutes les informations sont les bienvenues si elles sont disponibles, notamment la prédiction de gène et l'utilisation de cartes optiques. Pour ma part, je pense qu'il faut également ajouter des

résultats visuels afin de comparer au premier coup d'oeil et de pouvoir dicerner là où tous les scaffoldeurs se sont trompés par exemple. C'est pourquoi, dans l'article, nous avons ajouté un graphique de visualisation des scaffold : $scaffolds = f(génomederéférence)$ (voir Figure 22) sur lequel les erreurs d'assemblages comme les scaffolds longs apparaîssent clairement.

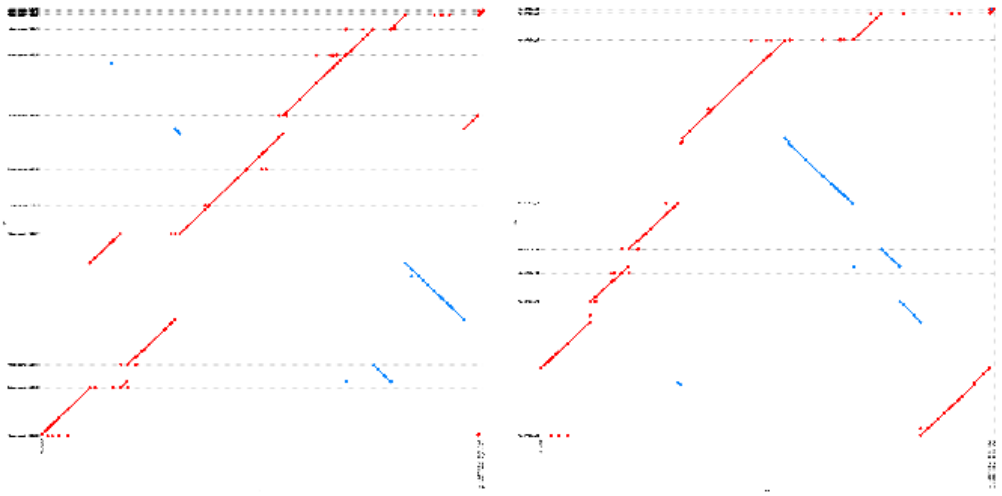

Figure 22: Tracé de l'alignement des contigs/scaffolds sur le génome : pour le génome Staphylococcus Aureus avec les contigs assemblé par Velvet et les scaffolds assemblés par Opera. En rouge les séquences présentant une inversion. Ce tracé permet à la fois d'avoir une vision globale des scaffolds et d'observer les erreurs d'assemblage.

4.1.6 L'évaluation avec QUAST

Afin de calculer toutes ces métriques et de permettre une évaluation rapide et précise, un seul logiciel suffit : QUAST (QUality ASsessment Tool for genome assemblies - outil d'évaluation de la qualité d'assemblage pour l'assemblage de génomes). QUAST est un outil automatique qui à partir des fichiers de scaffold (ou contigs) et du génome de référence calcule la plupart des métriques ainsi que tous les détails d'alignement et d'erreurs d'assemblage.

Ce fichier contient absolument tout le nécessaire pour pouvoir calculer différentes informations. Il m'a notamment permis de calculer une carte d'alignement des contigs sur le génome de référence en 4 niveaux :

1. le scaffold s'aligne en une fois sur la référence, on considèrera ces alignements comme sûrs,
2. le scaffold s'aligne à plusieurs endroits, c'est une répétition, cela nous permettra de voir comment le scaffoldeur a géré les répétitions,
3. des parties du scaffolds s'alignent sur la référence même s'il y a une erreur d'assemblage, cela peut nous permettre de savoir pourquoi le scaffoldeur a fait ces erreurs,
4. le scaffold a un autre alignement marqué sans signification car un autre alignement englobe la totalité du scaffold, ces alignements ne seront pas considérés.

A partir de cette carte j'avais pour projet de pouvoir vérifier l'assemblage en scaffold sur les 4 niveaux successivement. Cela m'a également permis de développer un script pour vérifier si deux contigs se chevauchent.
Dans un objectif d'être encore plus clairs, d'autres méthodes de visualisation doivent être développées comme il n'existe pas d'outil spécifique au scaffolding à ce jour.

4.2 visualisation de scaffoldeurs

La visualisation de scaffold n'est pas un domaine très développé (aucun outil dédié) bien qu'il existe beaucoup d'outils de visualisation en génomique. Mon travail pendant une partie du stage a été de chercher une solution de visualisation des scaffolds et de pourquoi pas adapter une technique existante à notre problème.

4.2.1 Genome View, Integrative Genomic Viewer (IGV)...

J'ai commencé par m'intéresser à certains logiciels permettant la visualisation du génome en 1 dimension. D'après leur site respectif :

- *"GenomeView est un navigateur et éditeur de génome autonome."*
- *"IGV est un outil de visualisation haute-performance pour une exploration intéractive de larges jeux de données de génomique. Il supporte une grande variété de type de données, dont les données de séquences de nouvelle-génération et les annotations génomiques."*

Certes ces logiciels permettent d'afficher le génome et même les lectures du génome en se basant sur fichier de mapping de ces lectures. Cependant, il ne permettent en aucune manière d'afficher des fichiers de scaffolds ou de contigs en les alignant à la référence, que ce soit un problème de format de fichiers acceptés ou tout simplement de software pas fait pour ça. Ainsi l'aspect intéressant d'une visualisation de scaffold, à savoir :

- visualisation simultanée de contigs, scaffolds et référence,
- visualisation d'informations telle que couverture par rapport aux paires de lectures ou encore couverture des scaffolds par des contigs,
- visualisation des erreurs d'assemblage,
- visualisation des espaces entre contigs dans les scaffolds.

J'ai donc du abandonner cette piste.

4.2.2 Circos

Circos est un logiciel de visualisation de données et d'informations. Il permet de visualiser les données de manière circulaires - Ce qui fait de Circos un outil idéal pour explorer des relations entre objets et positions, particulièrement en génomique. C'est vers lui que s'est penché mon choix et il constitue, de mon point de vue, la seule solution permettant à la fois d'afficher les chromosomes, les contigs, les scaffolds ainsi que leurs erreurs d'assemblage ou encore des informations comme la couverture, le tout en seul graphique. Il peut faire cela grâce aux différents types de données qu'il accepte.
Circos peut être automatisé. Il est controllé par un fichier de configuration complet en texte et des fichiers de données au format très simple (vois Annexe C pour plus de détails).
Mon projet vers le milieu du stage a été de créer un logiciel permettant de créer le fichier de configuration pour Circos ainsi que les fichiers de données. En voilà les spécifications :

1. Entrée :
   - fichiers de sortie de QUAST contenant les détails des erreurs d'assemblage pour chaque scaffold,
   - fichier contenant le génome de référence (format FASTA).
2. Sortie :
   - fichier de configuration pour Circos,
   - fichiers de données.
3. Ce logiciel doit être entièrement automatisé, impliquant lancement et génération de l'image par circos,
4. Ce logiciel doit permettre 2 types de tracés :
   - un montrant les scaffolds, contigs et le génome de référence en parallèle sur le même tracé,
   - un tracé coupé en deux avec d'un côté un scaffold précis et de l'autre la portion du génome de référence correspondante. Ce tracé permettra d'observer plus en détails les erreurs d'assemblage. Le logiciel doit permettre de choisir quels sont les scaffolds que l'on va tracer.
5. Optionnellement, on devrait pouvoir ajouter les informations telles que la couverture.
6. Dans l'idéal, on devrait pouvoir choisir les couleurs de chaque tracé.

J'ai pu développer la première partie du logiciel. Ainsi à l'heure actuelle :

- Il n'est pas entièrement automatisé,
- Il ne permet que la première sorte de tracé, voir Figure 23,
- Il ne permet ni courverture ni choix des couleurs.

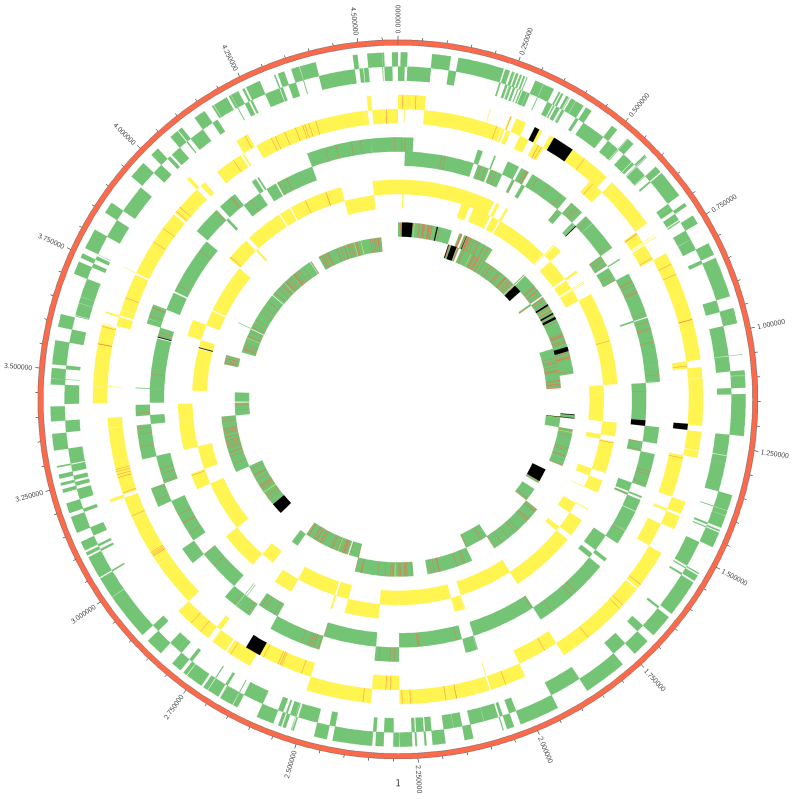

Figure 23: Tracé en parallèle des contigs et de 4 scaffolding par Sspace, Opera, Sopra et Scaftools (filtre à 10 liens).

4.3 Expérimentations

Pendant mon stage j'ai eu plusieurs sessions d'expérimentations à effectuer que ce soit :

- dans le cadre de notre collaboration avec le LIRMM :
  - Comparaison présentée à Montpellier,
  - Comparaison utilisée pour la rédaction d'un article,
  - Présentation de cette partie expérimentations en conférence à Lyon.
- pour l'étude des différentes techniques de scaffolding,
- pour étudier l'influence de la précision dans la détermination des tailles des espaces entre contigs,
- pour avoir une base avec laquelle comparer les essais de scaftools ou du scaffolder que j'ai développé.

4.3.1 Protocole

Softs :
Pour ces expérimentations, j'ai testé trois techniques de scaffolding :

- **Sopra**,
- **Sspace**,
- **Opera**,
- **Scaftools**.

Génomes :
Ces expérimentations ont été menées sur 6 génomes, 2 prokaryotes et 4 eukaryotes :

- **Staphylococcus Aureus** (alias staphylo),
- **Escherichia Coli** (alias ecoli),
- **Yersinia Pestis CO92** (alias ypco92),
- **Wolbachia endosymbiont** (alias wolbachia),
- **Homo Sapiens chromosome Y** (alias chrY),
- **Arabidopsis Thaliana** (alias arabido).

Jeux de données :
Les jeux de données de lectures ont plusieurs origines :

- Des jeux de données provenant de vrais séquençages pour staphylo, e. coli et arabido,
- Des jeux de données simulées pour Wolbachia, ypco92 et chrY.

Pour plus de détails sur les génomes et librairies utilisées, voir Annexe D.2.

Procédé : Pour effectuer ces expérimentations, j'ai suivi le même procédé pour chacun des génomes et chacun des scaffolders sauf scaftools afin de pouvoir les comparer de la manière a plus exacte possible. Voir figure 24 pour un schéma de ce procédé. J'ai donc commencé par créer les contigs à partir des librairies de lecture en utilisant Velvet. Puis, j'ai mappé les lectures aux contigs en utilisant bowtie. Ensuite, j'ai créé les scaffolds à partir de chacun des scaffolders pour enfin appliquer QUAST sur les résultats.
Pour le scaffolder scaftools, plusieurs filtres ont été testés concernant le nombre de paires de lectures minimum entre 2 contigs pour que le lien entre les deux soit pris en compte. 4 version de scaftools ont alors été testées avec le même procédé : sans filtre puis avec 3, 6 et 10 paires minimum.

Afin d'étudier l'effet de la précision dans la détermination de la taille des espaces entre contigs, j'ai remplacé tous les espaces dans les scaffolds produit par Opera par un certain nombre de N's : 0, 1 et enfin 100.

Les commandes utilisées pour lancer les scaffolders Sopra, Sspace et Opera faisaient rarement apparaître un changement par rapport aux paramètres par défaut de ceux-ci. Pour plus de détails, voir Annexe D.3.

Statistics	**Contigs**	**Opera**	**Sspace**	**Sopra**	**Scaftools**	**Scaftools 3**	**Scaftools 6**	**Scaftools 10**
# scaffolds	301	194	174	300	150	164	134	97
Longueur cumulée (bp)	2860307	2899757	2853630	2860279	2867857	2865655	2862009	2856206
# N's	0	39450	4001	0	7000	6250	5950	5700
NG50	48440	364718	48440	48440	895496	895353	895118	895118
NG50 corrigé	48149	197571	48149	48149	171910	171910	171910	171910
# erreurs d'assemblage	9	16	9	9	56	42	39	39
# relocations	9	16	9	9	53	41	38	37
# translocations	0	0	0	0	2	1	1	1
# inversions	0	0	0	0	1	0	0	1
# erreurs d'assemblage locales	8	81	8	8	74	73	72	69
Genome fraction (%)	97.726	97.714	97.73	97.726	97.889	97.841	97.847	97.769
# gènes trouvés (unique)	2720	2693	2721	2720	2721	2719	2710	2706
Temps d'exécution		33s	16s	1min40s	1s	<1s	<1s	<1s
Mémoire virtuelle		336M	497M	1,15G				

Table 1: analyse QUAST scaffolding pour le génome staphylo. On montre ici certaines statistiques pour l'assemblages :**# scaffolds :** le nombre de scaffolds dans l'assemblage. **Longueur cumulée :** la longueur cumulée des scaffolds (bp). **N's :** nombre total de base indécidées N (dans des espaces entre contigs le plus souvent) dans l'assemblage. **NG50 :** longueur maximum telle que la longueur cumulée des scaffolds de longueur supérieure ou égale produit au moins 50% de la longueur du génome de référence. **NG50 corrigé:** NG50 où la longueur des blocs alignés sont utilisés au lieu de la longueur des scaffolds, i.e. si un scaffold a une erreur d'assemblage, le scaffold est coupé à cet endroit. **# erreurs d'assemblage :** nombre de positions dans les scaffolds où 2 séquences collées sont séparées dans la référence (relocation); ou 2 séquences séparés dans la référence se chevauchent sur plus de 1kbp (relocation); ou 2 séquences collées sont sur différents brins dans la référence (inversion); ou 2 séquences collées sont sur différents chromosomes dans la référence (translocation). D'où **# relocations**, **# translocations**, **# inversions**. **erreurs d'assemblage locales :** nombre de relocations où 2 contigs devraient être collés mais sont séparés par moins de 1kbp ou se chevauchent sur moins de 1kbp. **Genome fraction (%) :** Nombre total de paires de base alignées dans la référence, divisée par la taille du génome. Une base est considérée comme alignée si il y a au moins un scaffold avec au moins un alignement sur cette base. Les scaffolds provenant de régions répétées peuvent apparaître plusieurs fois dans ce calcul. **# gènes trouvés (unique):** nombre de gènes uniques dans l'assemblage identifiés par GeneMark.hmm. **Temps d'exécution :** temps nécessaire au calcul des scaffolds. **Mémoire virtuelle :** Mémoire virtuelle nécessaire maximum pour calculer les scaffolds.

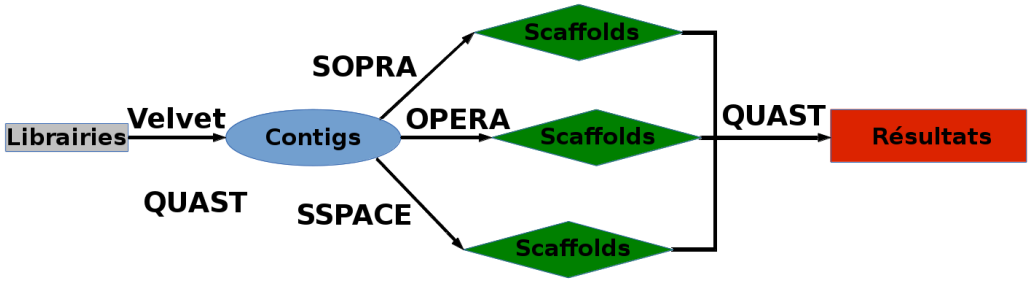

Figure 24: Protocole expérimentale pour les tests sur Sspace, Sopra, Scaftools et Opera.

4.3.2 Résultats

Les résultats de cette expérimentation pour le génome staphylo sont présentés dans le tableau 1 (pour les résultats des autres génome voir **??**) en gardant seulement les métriques préconisées dans la section précédente : Tableau 1.

4.3.3 Conclusions

Comparaison des scaffolds : Pour résumer les résultats sur les expérimentations de scaffolders :

- Au niveau du temps, Sspace est beaucoup plus rapide que Sopra et Opera mais scaftools les surclassent tous,
- Au niveau des capacités en mémoire nécessaires pour chaque scaffoldeur, les résultats semblent identiques,
- Ayant des NG50 plus grands et comparables, Sspace et scaftools sont plus précis que les deux autres,
- Pour scaftools, il reste cependant de gros problèmes, les scaffolds présentent en effet au moins 10 fois plus d'erreurs d'assemblage que les autres malgré le NG50 élevé. Y a-t-il un problème au niveau de la résolution qui implique des erreurs d'assemblage en bug,
- En résumé, scaftools semble être un scaffolder très aggressif et même peut être trop comme il semble faire beaucoup de liens chimériques. Je pense que cela peut venir de liens imaginaires effectués artificiellement avec des petits contigs d'où le NG50 corrigé qui reste bon. Il semble ainsi que ne pas contraindre le système à être connexe ne suffit pas à éviter ces liens artificiels. Sspace semble être un très bon compromis entre aggressivité d'assemblage, précision et surtout temps d'exécution. Sopra semble également être une très bonne solution, c'est pourquoi nous avons choisi de l'adapter (voir prochaine section).
- Certains résultats semblent vraiment étranges, à croire qu'il y a eu des erreurs pendant le processus d'assemblage :
  - Sur ChrY et Ypco92, il semble que Opera n'a rien fait,
  - Sur staphylo, c'est Sspace et Sopra,
  - Sur Wolbachia, Sspace, Sopra ET Opera.

Pour ce qui est de l'influence du nombre de N's, j'ai calculé un coefficient de corrélation sur chacune des métriques et presque aucune d'entre elle n'y était sensible. Les seules sur qui il semblait y avoir une influence étaient les NGx et LGx corrigés. Nous avons donc pu conseiller à l'équipe du LIRMM, qui n'avait pas le temps de mettre en place un système de détermination des espaces entre contigs, de mettre une longueur dépendant seulement du génome. Cette longueur nous l'avons déterminée en calculant une moyenne des tailles d'espace pour chaque génome.

De cette comparaison est ressortie surtout le sentiment qu'il manque encore des outils pour comparer les scaffolders comme il est assez difficile de choisir entre un scaffolder prudent qui ne fera que les choix sûr avec des scaffolds plus petits et un scaffolder plus aggressif qui fera le contraire.

4.4 Sopra

Sopra nous a paru un bon candidat pour trouver un procédé qui de base est efficace en comparaison aux autres scaffolders tout en permettant une intégration facile de cadres de résolutions exacts. Notre but va être d'étudier la faisabilité de cette intégration par rapport à la taille des domaines et à la complexité du problème.

4.4.1 Procédé de Sopra :

J'ai étudié le détail du code des différents fichiers de Sopra afin d'en comprendre le fonctionnement. D'abord, Sopra nécessite de lancer à la suite 3 scripts contenant le code pour :

- le parsing du fichier contenant les lectures mappées aux contigs,
- le pré-traitement des données qui en sont extraites.

Ainsi, pour résumer, pendant cette phases voilà les actions effectuées (pour plus de détails voir Annexe **??**) :

- récupération des données dans les fichiers de contigs et de mapping,
- calcul du nombre d'occurence des lectures, de la couverture des contigs, de la taille d'insert réel (empirique) des paires de lecture (grâce aux paires mappées à l'intérieur d'un même contig),
- filtrage des lectures (et des paires associées) qui ont un nombre d'occurence trop élevé,
- calcul du nombre de paire suggérant telle ou telle orientation relative entre les contigs ainsi que de la distance entre contigs suggérée.

Ensuite vient le script de scaffolding en lui-même : **perl s_scaf_v1.4.6.pl -w 2 -o [orientdistinfo_c5] -a [output_folder]**

1. **Pré-traitement :**
   - Filtrer le contigs trop petits, de trop grande couverture,
   - Filtrer les lectures apparaissant trop souvent,
   - Filtrer les arcs trop long, ambigüs ou soutenus par trop peu de paires,
2. **Scaffolding : orientation** : via un découpage du problème et une résolution des sous-problèmes par programmation dynamique ou recuit-simulé,
3. **Scaffolding : position** : via une modélisation en ressort qui mène à la résolution d'une équation linéaire par méthode du gradient conjugué (voir [ANNEXE]),
4. Répétition de ces deux étapes jusqu'à ce qu'il n'y ait plus de problème dans le système de ressorts,
5. **Scaffolding : découpage des scaffolds de haute densité :** via la résolution d'un problème de programmation linéaire en nombres entiers.

Ainsi on voit que Sopra fait le choix de séparer la détermination des orientations, position et découpage en tentant d'obtenir une solution optimale au sens du nombre de paires de lectures satisfaites. C'est cette résolution en plusieurs étapes que nous avons choisi. C'est ces dernières étapes : orientation, position et découpage des scaffolds que nous allons vouloir remplacer par des techniques de résolution exactes. Je n'ai eu le temps que de modéliser et tenter de résoudre les deux premières, examinons ces modélisations.

4.4.2 Définitions formelles

Comme vu plus haut, les liens de paires nous apportent 4 informations :

- les paires se fixent à un certain endroit des contigs et nous donnent ainsi leur orientation relative ou, pour utiliser le langage des contraintes, nous donnent une contrainte d'égalité ou d'inégalité entre les orientations des 2 contigs,
- l'orientation réelle de paires nous donne également la configuration exacte des contigs (voir Figure 25),
- l'orientation des deux lectures alliée à l'orientation assignée à un contig nous donne également la relation de précédence entre les contigs, l'ordre local (voir Tableau 2),
- en cas de conflit, le nombre de liens de paires indiquant telle ou telle orientation/ordre, nous donne une préférence si il y a une majorité,

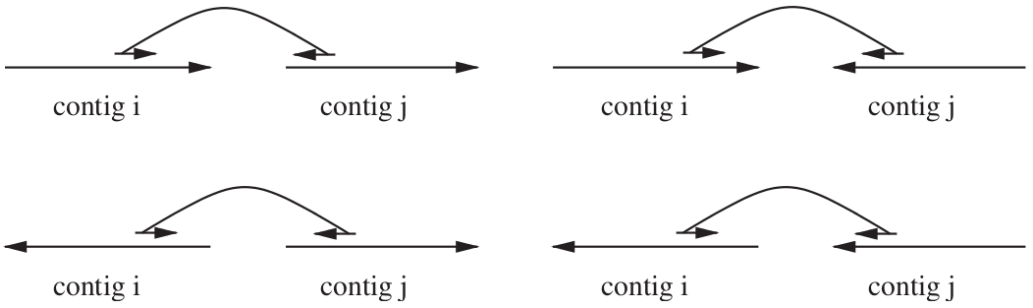

Figure 25: 4 configurations sont possibles avec l'orientation des lectures, en considérant les contigs i et j tels que le contig i précède le j.

orientation r_1	**orientation r_2**	**orientation C_1**	**orientation C_2**	**Ordre**
$\rightarrow$	$\rightarrow$	$\rightarrow$	$\leftarrow$	$C_1 \rightarrow C_2$
$\rightarrow$	$\rightarrow$	$\leftarrow$	$\rightarrow$	$C_2 \rightarrow C_1$
$\rightarrow$	$\leftarrow$	$\rightarrow$	$\rightarrow$	$C_1 \rightarrow C_2$
$\rightarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$C_2 \rightarrow C_1$
$\leftarrow$	$\rightarrow$	$\leftarrow$	$\leftarrow$	$C_1 \rightarrow C_2$
$\leftarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$C_2 \rightarrow C_1$
$\leftarrow$	$\leftarrow$	$\leftarrow$	$\rightarrow$	$C_1 \rightarrow C_2$
$\leftarrow$	$\leftarrow$	$\rightarrow$	$\leftarrow$	$C_2 \rightarrow C_1$

Table 2: Soit deux contigs C1 et C2 reliés par la paire de lecture r1 - r2. Le tableau donne pour toutes les situations possibles les orientations des lectures et contigs et l'ordre local induit. On obtient ainsi que C_1 précède C_2 ssi orientation r1 = orientation C_1 et C_2 précède C_1 ssi orientation r2 =< orientation C_2.

- la taille de l'espace entre les 2 séquences de la paire est connu avec une certaine incertitude, de là il n'y a qu'un pas vers la distance entre les 2 contigs.

Pour résumer, à ce stade, entre deux contigs reliés par des paires on a l'information d'orientation relative, d'ordre local et d'espacement théorique.
Pour la suite on désignera par c_i le contig i de longueur d_i (numérotation dans l'ordre de création) et on considére qu'il y a n contigs dans le problème :

- Chaque c_i a une orientation $o_i \in$ {F, R} (pour "Forward" et "Reverse"), un ordre $p_i \in$ {1,..., n} et une position $x_i \in$ {0,..., longueur du génome},
- En approximation, nous n'aurons d'informations entre 2 contigs que s'il existe une majorité de paires de lectures qui les relie et qui indique la même information d'orientation relative et d'ordre,
- Si cette condition est remplie pour 2 contigs c_i et c_j, l'ensemble des paires de lectures les reliant sera noté pair(c_i, c_j) et leur nombre est ||pair(c_i, c_j)||,
- Ces liens de paire nous indiquent une égalité ou une inégalité sur l'orientation des 2 contigs auxquels ils sont accrochés, nous noterons cette égalité $=_{paires}$ et l'inégalité $=_{paires}$. Ainsi si les liens de paire montrent une égalité d'orientation pour c_i et c_j, on aura $o_i =_{paires} o_j$, Les liens de paires donnent également une relation de précédence entre 2 contigs qui se traduit par une relation d'infériorité stricte entre les ordres des 2 contigs. On notera cette relation d'ordre local $<_{paires}$. Ainsi si les liens de paire montrent que ci est avant c_j, on aura $b_i <_{paires} b_j$,
- on note $J \in \mathbb{M}_n(\mathbb{R})$, la matrice d'adjacence représentant le graphe de contig après préprocessing. Soit 2 contigs c_i et c_j, J_{ij} contient donc le nombre de paires de lecture dans le groupe majoraitaire qui lit les deux contigs.

4.4.3 Orientation en WCSPs

A partir de ces définitions, et en se basant sur les contraintes d'égalités/inégalité, nous avons établi la modélisation en WCSP (voir Figure 26). Pour l'orientation, c'est le triplet **‹V, D, F›** :

- V = {o_1, ..., o_n} l'ensemble des orientations des contigs,
- D = {F, R}^{n} les ensembles associés,
- L'ensemble F est composé de fonctions d'arité 2, telles que :

$$\exists\ f_{ij}\ \in\ F\ ssi\ pair(c_i,\ c_j)\ \neq\ \emptyset$$

$$f_{ij}\ =\ \begin{cases} 0 & si\ o_i\ =\ o_j\ et\ o_i =_{paires}\ o_j \\ 0 & si\ o_i\ \neq\ o_j\ et\ o_i \neq_{paires}\ o_j \\ ||pair(c_i,\ c_j)|| & sinon \end{cases}$$

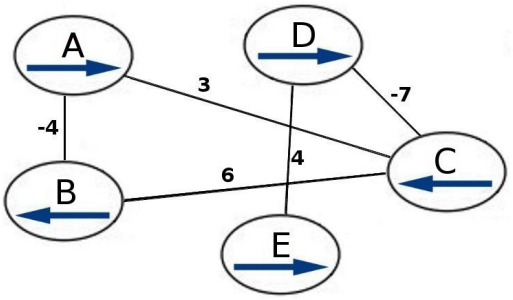

Figure 26: Modélisation en WCSPs d'un problème d'orientation pour un scaffolding de 5 contigs. Chaque noeud est un contig d'orientation de sens de la flèche. La pondération sur les arcs représente le nombre de paires de lecture reliant les deux contigs avec un signe positif si les contigs ont même orientations, négatif sinon.

4.4.4 Position en LP

Pour ce qui est de l'ordre, nous avons tenté de le modéliser également en WCSP mais nous sommes rapidement rendus compte que ce n'était pas viable à cause de la taille des domaines. Nous avons donc choisi de reprendre le modèle en ressorts de Sopra comme il permet à la fois d'obtenir l'ordre des contigs et l'espacement qui les sépare. Cependant, après une étude complète du code de cette partie et une tentative de réencodage en python, il s'est avéré que ce modèle contenait des parties étranges. Notamment, au lieu de se baser sur l'ordre local réel entre contig, Sopra intégrait dans ses calculs la précédence dans la numérotation des contigs. J'ai donc décidé de repartir de la définition même du modèle de ressort pour construire mon modèle d'ordre.

Soit un ensemble de ressort reliant les contigs R = {r_1, ..., r_m}, de raideurs K = {k_1, ..., k_m} avec ki $\in \mathbb{N}$ et longueurs à vide L = {l_1, ..., l_m} avec $l_i \in \mathbb{R}$,
Soit 2 contigs ci et cj liés par des paires de lectures modélisées par un ressort rq tels que $b_i < b_j$:

$$F_{c_i\ \rightarrow\ c_j}\ =\ -k_q * (x_j\ -\ x_i\ -\ l_q)\ (force\ appliquée\ sur c_j)$$
$$F_{c_j\ \rightarrow\ c_i}\ =\ -k_q * (x_i\ -\ x_j\ +\ l_q)\ (force\ appliquée\ sur c_i)$$

On voit ainsi que seul le signe de la longueur à vide considérée change. Soit deux contigs c_i et c_j, peut importe celui qui précède l'autre on a :

$$F_{c_i\ \rightarrow\ c_j}\ =\ -k_q * (x_j\ -\ x_i\ -\ \sigma_{ji} * l_q)\ avec\ \sigma_{ji}\ =\ signe(x_j - x_i)$$

Par bilan des forces :

$$\forall\ i\ \in\ \{1, ..., n\},\ \sum_{j=0,\ i,\ j\ liés\ par\ r_q}^{n} [-k_q * (x_j\ -\ x_i\ -\ \sigma_{ji} * l_q)]\ =\ 0$$

$$\Leftrightarrow\ \forall\ i\ \in\ \{1, ..., n\},\ -\sum_{j=0,\ i,\ j\ liés\ par\ r_q}^{n} (k_q * x_j) + \sum_{j=0,\ i,\ j\ liés\ par\ r_q}^{n} (k_q * x_i) + \sum_{j=0,\ i,\ j\ liés\ par\ r_q}^{n} (\sigma_{ji} * k_q * l_q)\ =\ 0$$

$$\Leftrightarrow \forall\, i \in \{1,...,n\}, \sum_{j=0,\ i,\ j\ liés\ par\ r_q}^{n} (k_q * x_j) - x_i * \sum_{j=0,\ i,\ j\ liés\ par\ r_q}^{n} k_q = \sum_{j=0,\ i,\ j\ liés\ par\ r_q}^{n} (\sigma_{ji} * k_q * l_q)$$

$$\Leftrightarrow AX = L\ avec : \begin{cases} A = \begin{cases} -\sum_{j=0,\ i,\ j\ liés\ par\ r_q}^{n} k_q & si\ i = j \\ k_q\ avec\ i,\ j\ liés\ par\ r_q & sinon \end{cases} \\ L = \sum_{j=0,\ i,\ j\ liés\ par\ r_q}^{n} \sigma_{ji} * k_q * l_q \end{cases}$$

On veut satisfaire un maximum de paires de lectures dans la solution, pour cela on pose la raideur comme égale à la valeur correspondant dans la matrice d'adjacence J :

$$\forall\, q \in \{1,...,m\} \ /\ r_q\ relie\ c_i\ et\ c_j,\ k_q = J_{ij}$$

Quand à la longueur à vide, regardons d'un peu plus près les 4 cas possibles pour 2 lectures pour 2 contigs c_i et c_j reliés par le ressort r_q et tels que $x_i < x_j$. Soit :

- insert_size : taille d'insert,
- read_length : longueur des lectures,
- start_r1 : position la plus à gauche de la lecture r_1 sur le contig c_i,
- start_r2 : idem avec r2 sur le contig c_j.

On obtient donc :

$$l_q = insert\_size + a + b + d_i\ avec : \begin{cases} a = \begin{cases} -d_i + start\_r1 & sio_i = F \\ -read\_length - start\_r1 & sinon \end{cases} \\ b = \begin{cases} -read\_length - start\_r2 & sio_j = F \\ -d_j + start\_r2 & sinon \end{cases} \end{cases}$$

Pour un exemple de graphe, voir Figure 27.

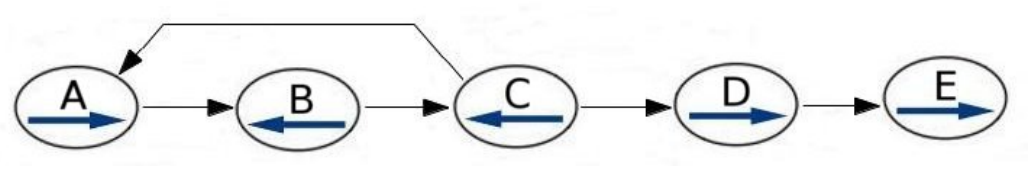

Figure 27: Modélisation d'un problème d'ordre pour un scaffolding de 5 contigs. Chaque noeud est un contig d'orientation de sens de la flèche. Le sens des flèches entre les contigs représente l'ordre local entre 2 contig.

Pour un exemple de la modélisation, voir Figure 28.

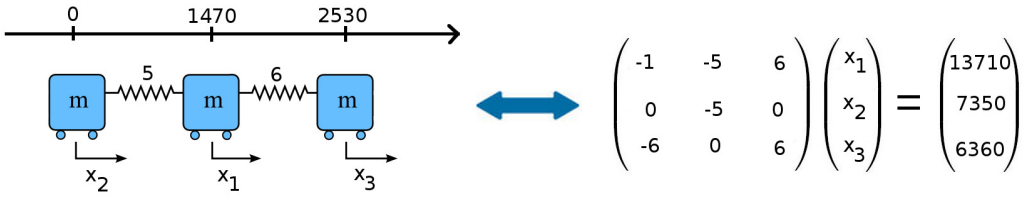

Figure 28: Modélisation d'un problème de ressorts à 3 contigs.

4.4.5 Programme python

Le programme python est construit en 4 modules :

1. Parsing,
2. Pre-traitement,
3. Orientation,

4. Position.

Pour modéliser le graphe de contig, j'ai utilisé la bibliothèque python **networkx**[3] qui m'a permis une manipulation de graphe facile et rapide.
Pour ce qui est de la résolution, j'ai dû utiliser la plateforme multi-paradigme Numberjack. C'est un outil de modélisation écrit en python pour la programmation par contrainte. Comme python bénéficie d'une communauté large et active, l'équipe de Numberjack dont mon tuteur Simon de Givry a choisi ce langage pour créer un outil parfait permettant d'intégrer la technologie CP dans des applications plus larges. Cet outil est rapide et possède des interfaces pour beaucoup de solveurs. Parmi ceux-ci, j'ai utilisé :

- pour l'orientation : Toulbar2, solveur spécialisé dans la résolution de WCSPs et développé par l'équipe du MIAT.
- pour la position : CPLEX, solveur polyvalent développé par IBM, particulièrement efficace dans la résolution de LP.

Pour voir le code l'application, se reporter à [ANNEXE]

4.4.6 Résultats

Les résultats obtenus sur l'orientation se sont avérés très encourageant en faveur de cette méthode exacte :

- Si la taille, et notamment la densité, du graphe représentant le problème à résoudre est trop importante, le problème n'est plus résolu,
- Mon tuteur a pu m'indiquer des paramètres repoussant cette limite :
  - **narycsp** : solveur de recherche locale INCOP,
  - **lds 4** : limited discrepancy search,
  - **elimDegree_preprocessing 3** : elimination des variables ayant seulement trois voisins au plus,
  - **preprocessTernaryRPC 128** : projection des triangles de 3 fonctions de couts binaires sur des triplets de 3 variables.
- Une solution satisfaisante (pour E. coli seulement 15 liens non respectés sur 500) est trouvée presque instantanément avec les même paramètres de filtre que Sopra,
- Cette solution est optimale.

Les résultats obtenus pour la position sont plus mitigés : comme notre solveur cherche une solution exacte au problème et qu'elle n'existe pas forcément, le solveur ne peut résoudre le problème. Cela est dû à la présence de cycles dans le graphe de contraintes qu'il faudrait éliminer. Le temps me manque pour finir cette tâche.

Je n'ai pas eu le temps de m'occuper de la dernière phase de Sopra : le découpage de scaffolds.

[3] `https://networkx.github.io/`

5 Conclusion

5.1 Etat du travail par rapport aux objectifs

Concernant les différents aspects que j'avais à traiter durant ce stage :

- Evaluation :
  - J'ai pu étudier les papiers les plus connus concernant la comparaison/évaluation de scaffolders,
  - J'ai également pu cataloguer toutes les métriques utilisées afin de pouvoir choisir les plus significatives,
  - Ces métriques on été utilisées pour le papier avec l'équipe du LIRMM sur scaftools.
- Visualisation :
  - J'ai pu examiner différentes solutions de visualisation même si certaines solutions que l'on m'a indiqué plus tard m'ont échappé : Narcisse, Ensembl, Mauve. Je n'aurais pas le temps de les étudier même si elles ont l'air de pouvoir permettre une visualisation dynamique en 1 dimension des données,
  - J'ai pu créer un script donnant en sortie les fichiers de données pour la première visualisation, permettant avec succès de comparer des scaffolds,
  - Il manque donc la deuxième visualisation et certaines options qui auraient permis plus de confort à l'utilisation mais c'est optionnel,
  - J'ai pu également indiquer pour le papier avec l'équipe du LIRMM la visualisation qui a été utilisée et semble claire.
- Expérimentations :
  - J'ai effectuer plusieurs phases d'expérimentations, apprenant à me servir des différents scaffolders (ce qui n'est pas peu faire dans le monde de la bioinformatique),
  - J'ai également pu présenté les résultats de ces expérimentations en utilisant les métriques trouvés en évaluation de scaffolders,
  - J'ai aussi profité de ces phases d'expérimentations afin de résoudre certaines questions qui se posaient à nous telles que l'importance de la précision dans l'espacement entre contigs,
  - Les résultats de ces expérimentations sont présents dna sl'article sur scaftools.
- SOPRA :
  - J'ai pu étudier le code complet de SOPRA (version avec contigs pré-fabriqués) que j'ai présenté dans un document à mon tuteur,
  - J'ai modélisé les modèles d'orientation et de positionnement en WCSP et en modèle de ressort,
  - Ces modèles ont été intégrés dans le processus,
  - Il s'est avéré que le modèle d'orientation était très efficace tant que l'on filtre assez le graphe de contigs (ce qui est de toute façon important puisque sinon les erreurs dans les données sont plus importantes),
  - Pour ce qui est du modèle de positionnement, des cycles s'étant glissés dans le graphe de contigs, il a été impossible de résoudre les problèmes à partir d'une certaine taille,
  - Je n'ai donc pas eu le temps de résoudre le problème concernant ce dernier modèle ni le temps d'implanter le découpage de scaffolds en fin de SOPRA,
  - Je n'ai pas non plus pu faire la modélisation du problème entier de scaffolding (orientation + positionnement) en un gros modèle MIP comme cela a été le cas dans [14].

5.2 Aller plus loin

Pour aller plus loin dans ce sujet, outre terminer le processus SOPRA, certaines pistes pourraient être envisagées :

- L'intégration de la couverture à chaque base des contigs dans le processus de scaffolding,
- Le prétraitement des lectures à l'aide d'un outil avant le scaffolding qui semble avoir montré des améliorations dans [16],

- L'utilisation de la programmation quadratique au lieu de linéaire pour la résolution du modèle en ressorts,
- La création d'une métrique pouvant évaluer plus précisément comment les répétitions sont gérées par les scaffolders,
- Trouver une fonction de coût permettant d'améliorer la prise en compte des petits contigs : à ce jour, on pénalise les petits contigs comme ils ont forcément moins de paires de lecture qui s'y fixent. Cette fonction de coûts dépendrait du nombre de paires se fixant sur le contig, de la taille du contig, de la taille d'insert et de la couverture moyenne du génome.

Deux pistes ont également été envisagées qui étaient l'objet de ma demande de bourse de thèse avec l'école MITT :

- se concentrer sur la décomposition en arbre de clusters de calcul du problème de départ comme c'est d'une importance cruciale dans la complexité de l'algorithme. Quelle est la meilleure stratégie de découpage ? Comment optimiser sa recherche ?
- exploiter les symétries du problème. En effet si on inverse totalement une solution des problèmes d'ordre et d'orientation, on obtient une nouvelle solution de même coût. L'idée serait alors d'utiliser cette propriété au sein d'une méthode d'optimisation prenant en compte l'arbre des clusters. En particulier, la symétrie pourrait améliorer le calcul du minorant pour l'algorithme DFBB, diminuant ainsi le temps de calcul. Une telle amélioration est assez générique pour être ensuite utilisée sur d'autres problèmes possédant la propriété de symétrie comme la coloration de graphe où l'échange des couleurs ne change absolument pas une solution.

5.3 Bilan personnel

Ce stage est en complète adéquation avec mon projet professionnel. Il m'aura tout d'abord permis d'avoir une première et forte expérience dans le milieu de la recherche. En effet, c'est dans ce milieu que je voudrais commencer une carrière pour à terme devenir enseignant-chercheur. Il m'aura également permis d'acquérir une double compétence en informatique et en biologie. L'aspect méthodologique m'a été apporté par le cadre des WCSP et de l'optimisation combinatoire en général tandis que j'ai exploré l'aspect biologique et pratique avec l'application à l'assemblage de génome et au scaffolding. J'ai également la chance de cottoyer au quotidien des acteurs de différents domaine qui m'apportent leur experience, dans leur domaine bien-sûr, mais surtout dans le domaine de la recherche en général. Ce stage me confirme dans mon ambition professionnel et je souhaite fortement poursuivre avec une thèse dans le milieu de la bioinformatique.

A Séquençage d'ADN

A.1 Principe et histoire

Le séquençage du génome consiste à déterminer l'ordre des nucléotides dans la molécules d'ADN et à le convertir en un fichier contenant l'information. Le processus de séquençage est crutial dans plusieurs domaines de la biologie et la médecine, ou plutôt le problème de déterminer le génome complet d'une manière robuste. La vitesse rapide atteinte avec les nouvelles technologies de séquençage a beaucoup augmentée comparée aux travaux précédents et permettent de séquencer complètement l'ADN de plusieurs espèces incluant le génome humain. Voici quelques points historiques à propos du séquençage d'ADN :

- début des années 1970 : premier séquençage d'ADN réalisé grâce à des techniques de chromatographie manuelles, chères, longues et laborieuses,
- 1977 : premier génome séquencé complètement (bacteriophage PhiX174).
- 1980 : les 2 premières techniques automatisées font gagner à leurs créateurs respectifs un prix Nobel en chimie. Parmi elles, la **méthode de séquençage Sanger** est toujours une des plus utilisées dans le monde (avec quelques améliorations bien sûr).
- fin des années 1990 : création des méthodes de séquençage de nouvelle génération ou séquençage haut débit.

A.2 Méthode de séquençage

A.2.1 Evolution :

Nous pouvons mentionner 3 étapes dans le développement des méthodes de séquençage au fil des ans.
D'abord, les méthodes de séquençage manuelles ont conduit à deux techniques automatisées :

- la méthode de Sanger ou de terminaison de chaîne,
- le séquençage de Maxam-Gilbert.

Grâce à ces techniques, le séquençage d'ADN est devenu vraiment faisable et le premier génome complet a été séquencé mais c'était toujours des méthodes lentes et onéreuses. Les premiers séquenceurs permettaient un séquençage de basse couverture avec des lectures relativement longues, de longueur pouvant aller jusqu'à 900 nucléotides.
C'est à ce moment là que les techniques ont intégré une phase de clonage de l'ADN et que les fragments obtenus du séquençage sont devenus plus courts et nombreux. C'est ce qui a donné naissance aux communément appelées **séquenceurs de nouvelle génération**. Depuis 2007, les séquenceurs de nouvelle génération ont augmenté de manièère significative la couverture du séquençage tout en offrant des lectures plus courtes (36 à 500 nucléotides). Parmi ces méthodess de nouvelle génération nous retrouvons le **454pyrosequencing**, le **Massively parallel signature sequencing (MPSS)** et le **séquençage Illumina (Solexa)**.

A.2.2 Techniques avancées :

"L'importante demande de séquençage à bas coût a conduit au développement du séquençage haut-débit (ou séquençage de nouvelle génération), technologies qui parallélisent le processus de séquençage, produisant des milliers et des millions de séquences. Les technologies de séquençage haut-débit ont pour but de réduire le coût du séquençage d'ADN au delà de ce qui est possible avec les méthodes standards.[...]"

Wikipedia - DNA sequencing, traduit de l'anglais

Sequençage shotgun :
La plupart des approches de séquençage utilisent une étape de clonage in vitro pour amplifier les molécules d'ADN individuelles, comme les méthodes de détection moléculaires ne sont pas assez sensibles pour le séquençage de molécules seules. Le séquençage shotgun est une méthode de séquençage faite pour analyser de larges séquences d'ADN (> 1000bp à un génome complet). l'ADN est coupé aléatoirement (shotgun) en de nombreux petits segments, qui sont séquencés
pour obtenir des lectures. Après avoir séquencé des fragments individuels, les séquences peuvent être assemblées.

Séquençage de novo :
Le **Séquençage de novo** se rapporte au séquençage d'ADN sans aucune connaissance préalable sur le

contenu. Les méthodes shotgun sont souvent utilisées pour séquencer des génomes larges mais c'est très complexe d'assembler les lectures et il y a souvent des trous dans la séquence finale. **A nouveau, il faut savoir qu'aucun génome (pas même celui de l'humain) n'a été déterminé avec 100% de certitude !**

A.3 Séquençage de Maxam-Gilbert et de Sanger

A.3.1 Séquençage de Maxam-Gilbert :

Le processus se fait en 6 étapes:

1. Purifier la séquence,
2. Ajouter un phosphate radioactif : La bordure de l'ADN (le lien externe entre les nucléotides) est composé de sucre et de phosphate. Le phosphate présent au début du fragment d'ADN est supprimé et rempacé par un phosphate radioactif,
3. Séparer les sous fragments : Une certaine enzyme est appliquée sur le fragment d'ADN marqué radioactivement. Cette enzyme, l'enzyme de restriction endonucléase, coupe l'ADN à une séquence spécifique (par exemple "AAGCTT") et nous obtenons un fragment avec une fin marquée radioactivement et une non-radioactive.
4. Identifier les bases : Quatre solutions sont produites et appliquées sur le fragment, chacune supprimant une base ou deux (respectivement G, G+A, C+T et C),
5. Fendre l'ADN : Les fragments sont coupés à l'endroit où les bases ont été supprimées,
6. Lire la séquence : Une technique qui lit les molécules radioactives sur un film à rayon X est utilisée pour détecter les fragments d'ADN séparés. En lisant le film à rayon X, les bandes de fragment d'ADN sont déterminées en se basant sur leur longueur dans chaque colonne séparée des quatre mixtures de réaction.

Voir la Figure 29, pour un résumé claire de cette méthode.

A.3.2 Séquençage de Sanger :

Réaction PCR :
Pour comprendre le séquençage de Sanger, d'abord il faut comprendre la réaction PCR (Polymerase Chain Reaction). C'est une version artificielle de la réplication de l'ADN, qui utilise une enzyme appelée DNA Polymérase pour rapidement amplifier l'ADN (voir Figure 30):

1. Commencer avec un fragment d'ADN standard à 2 hélices,
2. Le chauffer, pour casser le fragment en 2 brins séparés,
3. Pour lancer l'ADN polymerase, utiliser ce qui s'appelle des **primers** : de petits morceaux d'ADN synthétique qui se fixent au début du fragment d'ADN (ou, plus généralement, le bout d'ADN que l'on veut amplifier),
4. L'enzyme polymerase reconnaît ces primers une fois qu'ils sont fixés à l'ADN, et commence à remplir le reste de la séquence après le primer,
5. L'enzyme (représentée en bleu) attrape des dNTPs (nucléotides avec une bordure de sucre-phosphate) libres provenant de la solution tout autour, et les incorpore dans l'ADN. Elle sait quelle dNTP utiliser comme un seul nucléotide, le complément du nucléotide sur l'autre brin, peut se fixer à cet endroit.

Il existe des formes altérées des dNTPs, appelées **dideoxynucleotides (ddNTPs)**. Ce sont des dNTPs parfaitement fonctionnels, excepté le fait qu'ils leur manquent une partie de la bordure : en particulier, la partie nécessaire pour que le dNTP suivant puisse s'attacher. Ainsi, si nous avons des ddNTPs dans la solution quand on lance la réaction PCR, la réaction s'arrête quand un ddNTP est placé (voir Figure 31).

Méthode de Sanger :
Ce processus se fait en 4 étapes :

1. La région d'ADN à séquencer est amplifiée d'une certaine manière puis dénaturée pour produire de l'ADN à un brin.

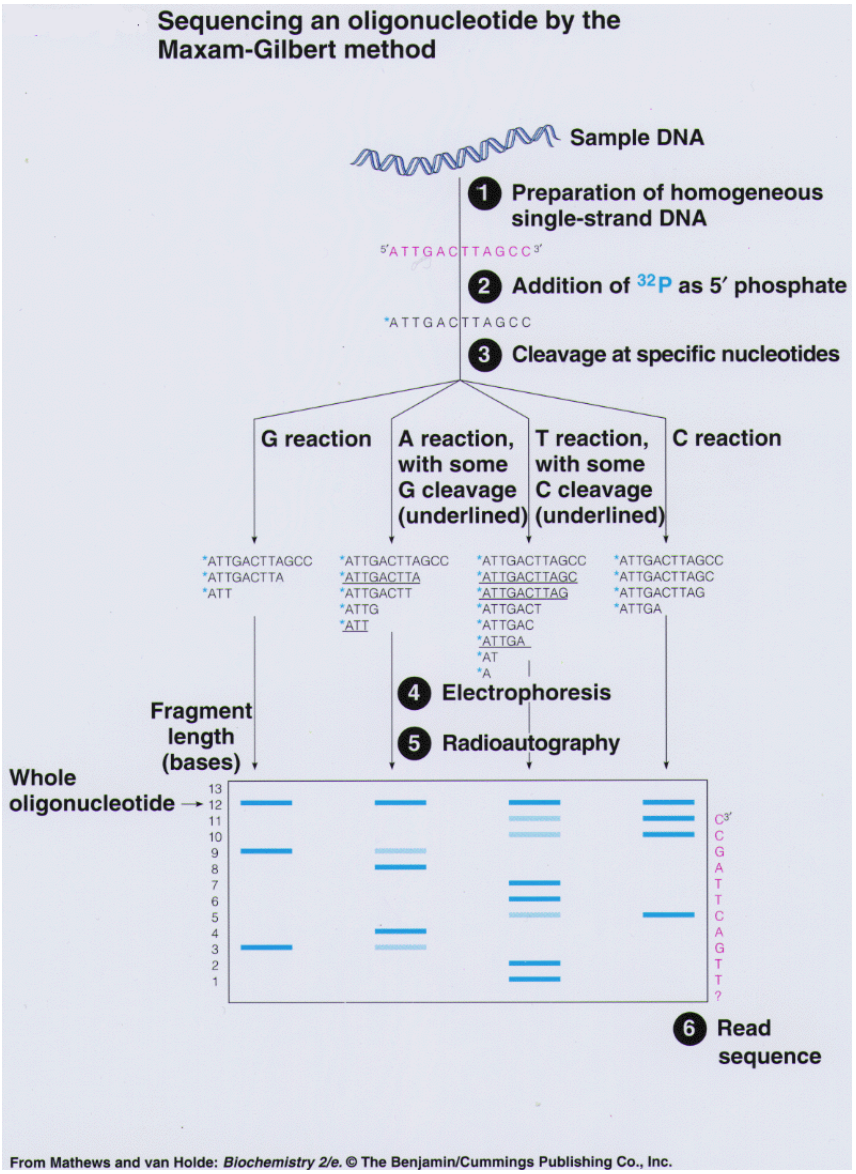

Figure 29: Séquençage de Maxam-Gilbert

2. un primer pour le séquençage est accroché à l'ADN à un brin.
3. la réaction PCR avec des ddNTPs est effectuée. En effectuant 4 réactions séparées, chacune contenant une petite quantité d'un des quatre ddNTPs en plus des 4 dNTPs, quatre ensembles séparés de fragments chain-terminated peuvent être produits.
4. chaque ddNTP est marquée avec un différent colorant pour qu'après la réaction PCR au lieu d'avoir juste un fragment d'ADN finissant par G, il y a des fragments rouges finissant par G, des bleus finissant par T etc... Maintenant, il suffit de faire descendre l'ADN le long d'un tube capillaire. Les fragments les plus petits bougent plus vite, les plus long plus lentement et la couleur des fragments d'ADN peuvent être lues l'une après l'autre quand elles sortent pour déterminer la séquence.

Voir Figure 32, pour un résumé claire de cette méthode.

B Assemblage

B.1 Principe et histoire

L'assemblage de séquence est le processus consistant à aligner, orienter et fusionner les fragments d'ADN obtenus durant la phase de séquençage. Cette étape est nécessaire parce que le séquençage d'ADN ne peut pas

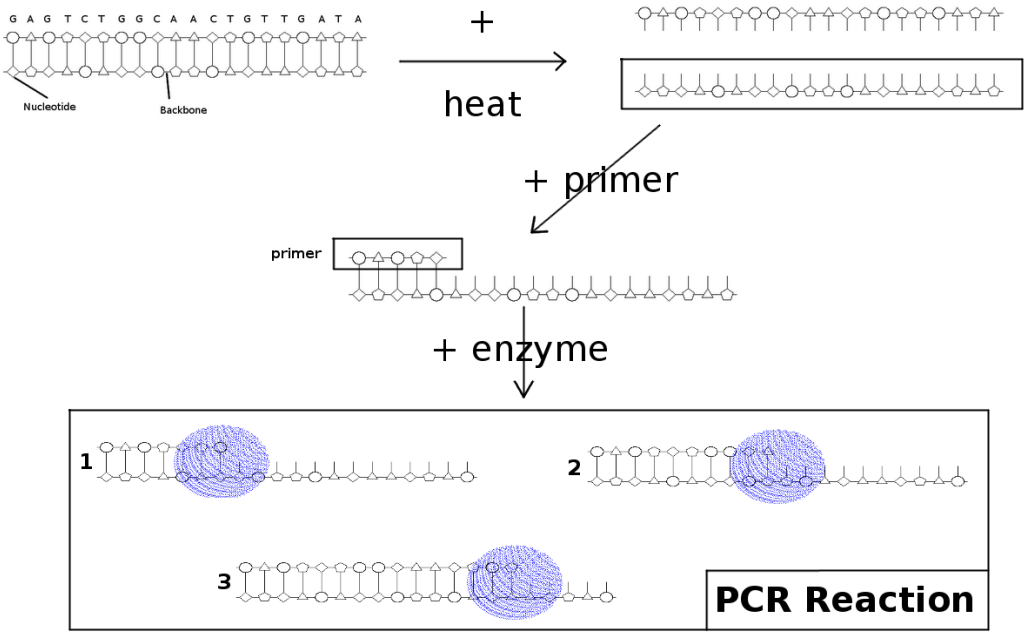

Figure 30: Réaction PCR

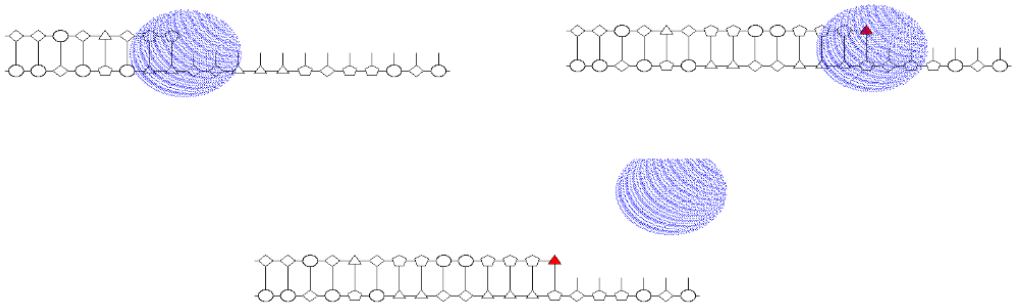

Figure 31: Terminaison de chaîne

retourner en sortie le génome complet en une lecture. Au ieu de cela, on obtient de petites séquences de 20bp à 40000bp (selon la technologie utilisée) en utilisant souvent des fragments venant de séquençage shotgun. A partir des lectures de la phase de séquençage, l'assembleur va essayer de mettre les lectures ensemble en se basant sur leur chevauchement. L'assemblage est généralement fait en 2 étapes :

- D'abord, les fragments sont assemblés en utilisant le chevauchement des lectures pour construire de plus grandes séquences complètes appelées contigs. L'information utilisée est ainsi généralement limitée aux lectures simple (sans l'information de paire).
- Ensuite vient la phase de scaffolding : les contigs sont ordonnés, c'est à dire qu'on leur donne une orientation, un ordre et que l'on estime la taille des espaces entre ces contigs. chacune de ces séquence est appelée scaffold.

Dans ces deux phases, les séquences (respectivement contigs) sont soit alignés en utilisant un algorithme glouton, soit (plus souvent) mises dans un graphe. Pour l'assemblage en contigs, une séquence valide sera alors un chemin dans le graphe et un scaffold une partie connexe du graphe. Nous examinerons les différentes sortes de graphe utilisées dans les prochaines sections. Pendant le scaffolding, nous avons une séquence de contigs (quelque fois avec des trous) et alors se pose une question : cette séquence est-elle proche (ou identique) à la séquence biologiquement vraie ? Alors se pose 2 choix possibles : soit nous avons déjà une séquence à laquelle se comparer, soit on détermine la séquence à partir de rien, c'est appelé le séquençage de novo.

Voici quelques faits historiques à propos de l'assemblage d'ADN :

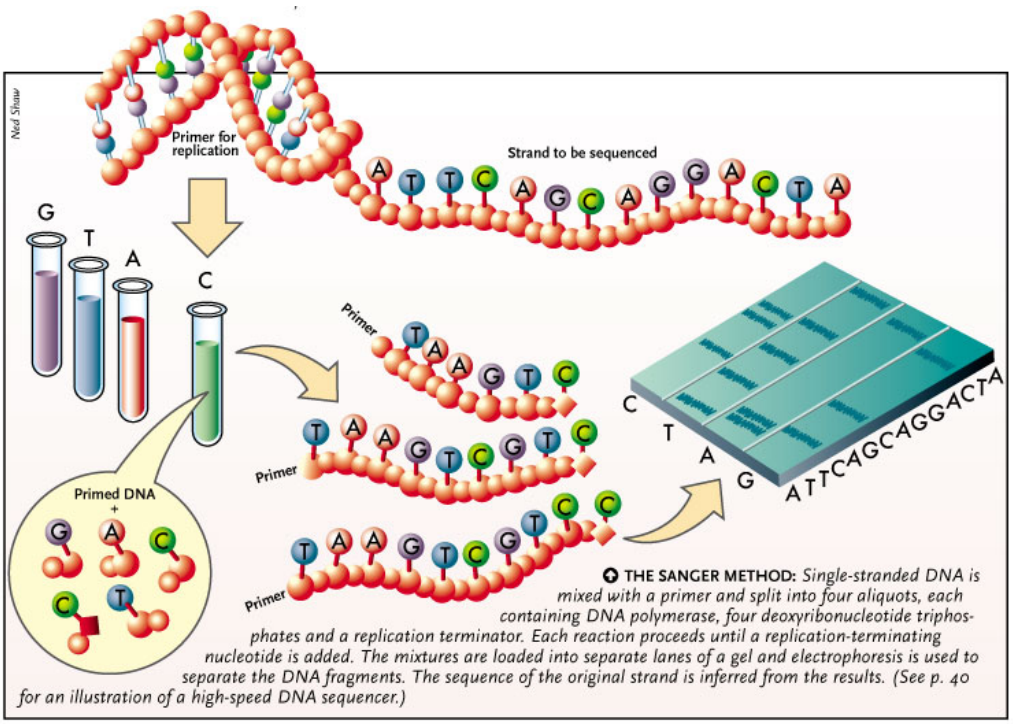

Figure 32: Séquençage de Sanger

- Les débuts du séquençage d'ADN : les scientifiques pouvaient seulement obtenir quelques séquences de courte longueur (quelques douzaines de paires de base) après des semaines de travail en laboratoire. Aussi, ces séquences pouvaient être alignées en quelques minutes à la main,
- 2004/2005 : pyrosequencing créé. Cette nouvelle méthode de séquençage génére des lectures plus courtes qu'avec le séquençage de Sanger : avant 100bp, maintenant 400-500bp. Son débit bien plus grand et à moindre coût (comparé au séquançage de Sanger) a poussé à l'adoption de cette technologie par les centres de génomique, qui à leur tour ont approfondi le développement d'assembleurs pouvant gérer les librairies de lectures,
- 2006 : la technologie Illumina (previously Solexa) est disponible et peut générer 100 millions de lectures par utilisation sur un séquenceur. A comparer avec les 35 millions de lectures du projet Genome Humain qui a nécessité des années pour produire le génome sur des centaines de séquenceurs. Illumina était initialement limité à une longueur de 36bp, la rendant moins appropriée pour de l'assemblage de novo, mais de nouvelles versions de la technologie sont capables d'obtenir des longueurs de lecture de plus de 100bp avec un insert de 300-400bp. Les lectures provenant de la technologie Illumina sont maintenant très utilisées dans les méthodes d'assemblage (comme celles étudiées ici).

B.2 Modèles et techniques de contigage

B.2.1 Shortest common superstring (SCS ou plus petite super-string commune) :

L'assemblage de génome était d'abord vu comme une instance du problème **SCS** :

- Soit s une super-string d'un ensemble de chaînes de caractères (string) S = $\{s_1$, ..., $s_n\}$. Si chaque s_i est une sous-string de s, et s satisfait la condition "pas de caractère en plus", i.e. pour chaque i $\in$ [1, |s|], au moins un élément de S est **aligné** avec s (est égal à une sous-string de s) à un intervalle contenant i.
- La SCS d'un ensemble est alors la (ou une) string la plus courte de cet ensemble.

On pensait que la SCS des séquences obtenues du séquenceur était la solution du problème d'assemblage mais il y a 3 problèmes que cette modélisation ne pouvait résoudre :

1. Tandem repeats collapse : "ARRRRRB" est assemblé en "ARRB" avec des lectures de longueur 2, effondrant les répétitions de R,

2. Over-collapsing: "ARBRCRD" est assemblé en "ARBSCSD", où S = R[1..r] + R[|R|-r..] avec r la longueur de lecture. Chaque lettre est une région plus grande que la longueur de lecture et S est aussi une séquence valable,

3. ce problème de trouver la SCS est NP-Difficile et Max-SNP Difficile - aucun algorithme ne peut trouver une approximation du problème en un temps polynomial. En fait c'est 4OPT max i.e. qu'on peut toujours trouver une string au maximum 4 fois plus grande que la plus petite string.

Pour passer outre ces problèmes, des modèles d'assemblage qui s'attaquent à ces problèmes précis ont été créés : graphes de string et de graphe de De Bruijn.

B.2.2 Graphes de string :

Les graphes de string sont une représentation naturelle des chevauchements d'un ensemble de string :

- On considère une définition très simple d'un chevauchement entre deux string. 2 string (r, r') sont dites k-chevauchantes si le suffixe de r est égal exactement au préfixe de r sur une longueur fixée de k caractères, où un préfixe de s est s[1..i] avec $1 \leq i \leq |s|$ et un suffixe est s[j..|s|] avec $1 < j < |s|$,

- Pour un ensemble de strings S, le graphe de chevauchement est un graphe orienté. Ses sommets sont S, et un arc $s_1 \rightarrow s_2$ existe si il y a un k-chevauchement entre s_1 et s_2,

- Le graphe de string est un graphe orienté dont les sommets et les arcs sont inclus dans ceux du graphe de chevauchement:

  - Inclusion de lectures (lectures qui sont incluses dans les autres lectures),
  - arc transitivité (arcs de la forme $s_1 \rightarrow s_3$, dès qu'il existe s_2 tel que $s_1 \rightarrow s_2$ et $s_2 \rightarrow s_3$) sont supprimés.

  Voir Figure 33, pour un exemple de graphe de string avec un ensemble de string abcd, bcde, cdef, defi, efic, ficd, icde, defg, efgh et des 3-chevauchements.

- Le problème d'assemblage est alors une généralisation du chemin Hamiltonien : trouver un chemin de longueur minimum visitant chaque sommet du graphe de string une fois.

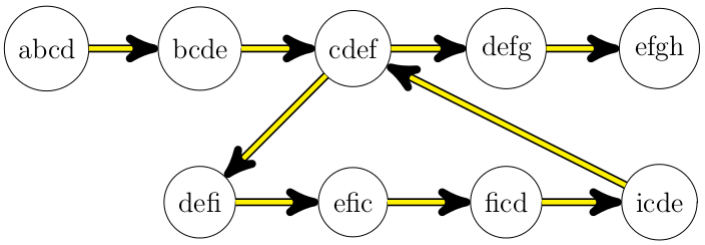

Figure 33: Graphe de string

Le problème est NP-Difficile (réduction du problème SCS). Si c'est le cas, c'est seulement à cause de toutes les répétitions dans la séquence d'ADN. Sans ces répétitions, une solution pourrait être trouvée en un temps polynomial comme une instance du problème du postier chinois.
Parmi les scaffoldeurs récents les plus utilisés, celui qui utilise les graphes de string est SGA [5].

B.2.3 Graphes de De Bruijn :

Un graphe de De Bruijn est un graphe spécialement créé pour représenter des chevauchements entre séquences de symboles et est donc très pertinant pour le problème d'assemblage de séquence :

- Pour un ensemble de string S, le graphe de De Bruijn est un graphe orienté dont les sommets sont toutes les sous-string de longueur k de chaque élément de S, et un arc $s_1 \rightarrow s_2$ est présent si il y a un (k - 1)-chevauchement entre s_1 et s_2. Voir figure 34 pour un exemple d'un graphe de De Bruijn avec l'nesemble de string abcd, bcde, cdef, defi, efic, ficd, icde, defg, efgh et des string de longueur k = 3

- Si s est une string, un **walk** w(s) est un chemin dans le graphe de la forme s[1..k] → s[2..k + 1] → ... → s[|s| - k + 1..|s|],

- Pour un chemin w', un **sous-chemin** w de w' est un chemin sur un sous-ensemble des sommets et des arcs de w. On dit que w est un **sous-walk** de w' si w est un sous-chemin de w'. On observe que tout chemin w du graph est un walk w = w(s) pour une string s,
- Un **super-walk** pour un ensemble de string S est un walk w' telle que pour toute s ∈ S, w(s) est un sous-walk de w',
- Le problème de l'assemblage pour ce modèle est connu comme le **de Bruijn Superwalk Problem (BSP ou problème du super-walk de De Bruijn)** : pour un ensemble de string S = s_1, ..., s_n sur un alphabet Σ et un entier k, est-ce qu'il existe un super-walk de longueur minimum pour S dans le graphe de De Bruijn correspondant ?

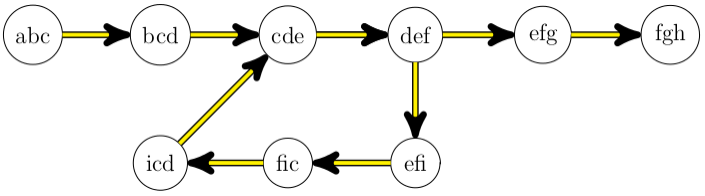

Figure 34: Graphe de De Bruijn

Ce problème est NP-Difficile pour |Σ| ≥ 3 (ce qui est le cas pour le séquençage d'ADN) et tout k ≥ 1 (aussi le cas avec des séquences de longueur 1 n'est pas informatif).
Ces scaffoldeurs utilisent les graphes de De Bruijn :

- PE-assembler [22],
- velvet [6],
- ABySS [23],
- meraculous [24],
- Monument (INRIA, chikhi) [25],
- celera [26],
- SOAPdenovo [27],
- ALLPATH-LG [28],
- Minia [7].

C Circos

Circos est un logiciel de visualisation de données et d'informations. Il est controllé par un fichier de configuration complet en texte et des fichiers de données au format très simple (vois Annexe **??** pour plus de détails):

1. Fichier karyotype : définit les chromosomes ou les scaffolds selon ce que l'on veut afficher. Le karyotype marche comme une base sur laquelle le reste des tracés s'aligne. Chaque chromosome a un nom, un label, une position de début et de fin et une couleur :
   - chr - hs1 1 0 249250621 chr1
   - chr - hs2 2 0 243199373 chr2
   - ...
2. line, scatter, histogram, heat map : les tracés sont des tracés en 2D qui associent une valeur à une position dans le génome. Ces tracés pourront être utilisés pour visualiser des informations comme la couverture :
   - *#chr start end value [options]*
   - hs5 50 75 0.75

3. tile : définit un intervalle sur le même chhromosome. C'est utilisé pour visualiser les pistes en parallèle du génome. On peut y ajouter des couleurs par intervalles, affichant ainsi les erreurs d'assemblage à souhait :
   - *#chr start end [options]*
   - hs5 50 75
4. links : associe 2 intervalles entre le même ou différents chromosomes et le trace comme une ligne ou des bandes :
   - *# chr1 start1 end1 chr2 start2 end2 [options]*
   - hs1 200 300 hs10 1100 1300

D Expérimentations

D.1 Résultats

Les tableaux 3, 4, 5, 6, 7 présentent les résultats des expérimentations respectivement pour les génomes chrY, e. coli, wolbachia, ycpo92 et arabido.

D.2 Données :

D.2.1 Génomes :

Ces expérimentations ont été menées sur 6 génomes, 2 prokaryotes et 4 eukaryotes :

- **Staphylococcus Aureus** (alias staphylo), NC_010079 (Staphylococcus aureus sous-espèce aureus USA300_TCH1516, génome complet),
- **Escherichia Coli** (alias ecoli), U00096.2 (Escherichia coli K-12 MG1655, génome complet),
- **Yersinia Pestis CO92** (alias ypco92), NC_003143.1 (Yersinia pestis CO92 chromosome),
- **Wolbachia endosymbiont** (alias wolbachia), NC_010981.1 (Wolbachia endosymbiont de Culex quinquefasciatus Pel chromosome, génome complet),
- **Homo Sapiens chromosome Y** (alias chrY), human_g1k_v37 GRCh37,
- **Arabidopsis Thaliana** (alias arabido), TAIR10.

D.2.2 Librairies :

Les jeux de données de lectures ont plusieurs origines :

- Des jeux de données provenant de vrais séquençages :
  - Staphylococcus Aureus, librairies de short jump (du jeu de données de GAGE [20]),
  - Escherichia Coli, librairie de lectures Illumina SRR001665,
  - Arabidopsis Thaliana, librairie de lectures Illumina SRR616966.
- Des jeux de données simulées :
  - Wolbachia, simulés avec toyseq, un outil spécifique développé pour le Variathon [4]. Ce jeu de données présente une fréquence de variantes mineures de 20%,
  - Yersinia Pestis et le chromosome Y du génome humain, simulé avec wgsim, un simulateur de lectures classique [5].

[4] `http://bioinf.dimi.uniud.it/variathon`
[5] `https://github.com/lh3/wgsim`

Statistics	Contigs	Opera	Sspace	Sopra	Scaftools	Scaftools 6	Scaftools 10
# scaffolds	24560	24560	11867	13898	5643	6618	5785
Longueur cumulée (bp)	14210800	14210800	13795906	14851307	17050107	10567926	9613796
# N's	0	0	405187	749055	1011400	389250	336300
L50	1472	1472	782	485	291	819	762
L50 corrigé	2057	2057	4348	7180	10203	3147	3233
# erreurs d'assemblage	0	0	1	31	630	305	132
# relocations	0	0	1	31	630	305	132
# translocations	0	0	0	0	0	0	0
# inversions	0	0	0	0	0	0	0
# erreurs d'assemblage locales	0	0	2289	3824	12653	5542	4904
Genome fraction (%)	17.374	17.374	19.447	20.439	20.819	15.106	13.889
# gènes trouvés (unique)	3745	3745	3348	2978	2126	2606	2400
Temps d'exécution	NA	4min27	3min45s	10min54s	7min35.7818s	6min52s15	6min3s62
Mémoire virtuelle	NA	262.566M	512.605M	1.718G	NA	NA	NA

Table 3: analyse QUAST scaffolding pour le génome chrY. On montre ici certaines statistiques pour l'assemblages :**# scaffolds :** le nombre de scaffolds dans l'assemblage. **Longueur cumulée :** la longueur cumulée des scaffolds (bp). **N's :** nombre total de base indécidées N (dans des espaces entre contigs le plus souvent) dans l'assemblage. **NG50 :** longueur maximum telle que la longueur cumulée des scaffolds de longueur supérieure ou égale produit au moins 50% de la longueur du génome de référence. **NG50 corrigé:** NG50 où la longueur des blocs alignés sont utilisés au lieu de la longueur des scaffolds, i.e. si un scaffold a une erreur d'assemblage, le scaffold est coupé à cet endroit. **# erreurs d'assemblage :** nombre de positions dans les scaffolds où 2 séquences collées sont séparées dans la référence (relocation); ou 2 séquences séparés dans la référence se chevauchent sur plus de 1kbp (relocation); ou 2 séquences collées sont sur différents brins dans la référence (inversion); ou 2 séquences collées sont sur différents chromosomes dans la référence (translocation). D'où **# relocations**, **# translocations**, **# inversions**. **erreurs d'assemblage locales :** nombre de relocations où 2 contigs devraient être collés mais sont séparés par moins de 1kbp ou se chevauchent sur moins de 1kbp. **Genome fraction (%) :** Nombre total de paires de base alignées dans la référence, divisée par la taille du génome. Une base est considérée comme alignée si il y a au moins un scaffold avec au moins un alignement sur cette base. Les scaffolds provenant de régions répétées peuvent apparaître plusieurs fois dans ce calcul. **# gènes trouvés (unique):** nombre de gènes uniques dans l'assemblage identifiés par GeneMark.hmm. **Temps d'exécution :** temps nécessaire au calcul des scaffolds. **Mémoire virtuelle :** Mémoire virtuelle nécessaire maximum pour calculer les scaffolds.

Statistics	Contigs	Opera	Sspace	Sopra	Scaftools	Scaftools 3	Scaftools 6	Scaftools 10
# scaffolds	866	550	121	495	116	139	145	149
Longueur cumulée (bp)	4567519	4575931	4535476	4576535	4605019	4603869	4603569	4603369
# N's	0	8412	4781	12750	36100	35150	34900	34700
NG50	16538	82905	82307	82605	183075	139306	139306	139306
NG50 corrigé	16538	80535	82307	82405	132640	134887	134887	132813
# erreurs d'assemblage	0	4	1	5	57	34	35	34
# relocations	0	4	1	5	57	34	35	34
# translocations	0	0	0	0	0	0	0	0
# inversions	0	0	0	0	0	0	0	0
# erreurs d'assemblage locales	0	106	38	103	415	419	419	417
Genome fraction (%)	97.189	97.268	97.631	97.513	97.749	97.746	97.754	97.754
# gènes trouvés (unique)	4378	4296	4292	4316	4407	4413	4411	4417
Temps d'exécution	NA	(CPU) 3h28min16s	16min4s	39min48s	12.74258s	0.58s	6.645s	7.835s
Mémoire virtuelle	NA	10.745G	1.453G	16.878G	NA	NA	NA	NA

Table 4: analyse QUAST scaffolding pour le génome e. coli.

Statistics	Contigs	Opera	Sspace	Sopra	Scaftools	Scaftools 3	Scaftools 6	Scaftools 10
# scaffolds	280	277	225	243	79	101	91	73
Longueur cumulée (bp)	1316232	1316336	1310665	1315774	1325527	996873	956436	611545
# N's	2776	2880	2841	2956	12698	8436	7336	5212
NG50	17136	17136	17372	17372	31212	13031	12956	17340
NG50 corrigé	16194	16194	16194	16194	18121	9289	9289	24
# erreurs d'assemblage	6	7	7	9	86	54	41	26
# relocations	6	7	7	9	86	54	41	26
# translocations	0	0	0	0	0	0	0	0
# inversions	0	0	0	0	0	0	0	0
# erreurs d'assemblage locales	14	14	13	15	56	40	40	31
Genome fraction (%)	86.628	86.631	86.765	87.034	86.67	65.607	63.21	40.162
# gènes trouvés (unique)	1292	1292	1301	1298	1368	1036	978	978
Temps d'exécution	NA	13s	7s	3s	0.647924s	0.16s	0.18s	0.5s
Mémoire virtuelle	NA	251.273M	447.754M	229.652M	NA	NA	NA	NA

Table 5: analyse QUAST scaffolding pour le génome wolbachia.

Statistics	Contigs	Opera	Sspace	Sopra	Scaftools	Scaftools 6	Scaftools 10
# scaffolds	4983	4983	2508	1336	219	210	205
Longueur cumulée (bp)	4291811	4291811	4246657	4288356	4400456	4329597	4301178
# N's	0	0	28632	62115	54700	53500	52600
NG50	1207	1207	3034	8335	43012	38059	38059
NG50 corrigé	1206	1206	2997	8066	37054	36354	36354
# erreurs d'assemblage	0	0	2	3	24	13	9
# relocations	0	0	2	3	24	13	9
# translocations	0	0	0	0	0	0	0
# inversions	0	0	0	0	0	0	0
# erreurs d'assemblage locales	0	0	262	450	750	742	734
Genome fraction (%)	78.369	78.369	84.571	88.035	92.752	91.327	90.77
# gènes trouvés (unique)	5397	5397	4968	4669	4458	4387	4347
Temps d'exécution	NA	39s	44s	6min26s	5.55s	3.81s	3.42s
Mémoire virtuelle	NA	344.809M	462.016M	684.727M	NA	NA	NA

Table 6: analyse QUAST scaffolding pour le génome ypco92.

istics	Contigs	Opera	Sspace	Sopra	Scaftools	Scaftools 3	Scaftools 6	Scaftools 10
affolds	172616	156036	28979	74793	39319	22995	16257	12370
umulée (bp)	115728669	116339090	107736992	115462916	112819169	108091684	104249755	99689219
N's	0	610421	1877741	1570050	3063650	2624350	2253100	1879600
G50	48440	364718	48440	48440	895496	895353	895118	895118
corrigé	4338	13249	19616	13520	26384	23505	20830	17741
'assemblage	12	25	121	204	3745	1791	1044	542
cations	6	14	67	147	1441	843	523	305
locations	6	10	53	57	2301	946	521	236
ersions	0	1	1	0	3	2	0	1
semblage locales	18	6997	8872	10199	38939	34894	30869	26843
raction (%)	79.111	79.15	83.541	83.646	83.759	82.34	80.896	78.793
uvés (unique)	35993	30080	30025	31262	28835	28454	28098	27499
'exécution	NA	(CPU) 22h11min13s	3h25min42s	31h3min24s	8h13min42.674s	5h24min24s55	6h3min8s83	4h22min57s37
virtuelle	NA	15.167G	9.355G	25.169G	NA	NA	NA	NA

Table 7: analyse QUAST scaffolding pour le génome arabido.

D.3 Utilisation des scaffolders

Les commandes utilisées pour lancer les scaffolders Sopra, Sspace et Opera :

- **Velveth :**
  velveth [output] 31 -fastq -short [reads_file1.fastq] [reads_file2.fastq]
  avec : **[output]:** nom du dossier pour les fichiers de sortie, **31:** la longueur de k-mer pour le graphe de De Bruijn, **-fastq:** format de fichier pour les librairies, **-short:** les lectures en entrée sont des paired-ends, **[reads_file1.fastq] and [reads_file2.fastq]:** fichiers pairés des paires de lecture.

  velvetg [output] -ins_length [insert size] -amos_file yes -read_trkg yes -cov_cutoff 5
  avec : **[output]:** nom du dossier pour les fichiers de sortie, **-ins_length:** taille d'insert, **-amos_file:** exporte l'assemblage dans un fichier AMOS, **-read_trkg:** pistage des positions de lectures courtes dans l'assemblage, **-cov_cutoff:** couverture minimum pour suppression de noeuds de faible couverture.

- **Bowtie :**

  - Pour Sopra et Opera :
    bowtie-build [contigs.fasta] [output] (Sopra, Opera)
    avec : **[contigs.fasta]:** fichier de contigs en entrée, **[ebwt_outfile_base]:** écrit les données Ebwt dans des fichiers avec ce dir/basename.

    bowtie -v 0 -m 1 -f –sam [contigs.fasta] [reads_file.fasta] (Sopra)
    bowtie -v 3 -a -m 1 -S -t [contigs.fasta] -p 15 [reads_file.fasta] (Opera)
    avec : **-v:** nombre maximum d'espaces permis pendan le mapping avec bowtie, **-a:** rapporte tous les alignements par lecture, **-m:** nombre maximum d'alignements pour une lecture avant suppression, **-f:** les fichiers de requête en entrée sont au format (multi-)FASTA .fa/.mfa, **–sam/-S:** écrits les résultats au format SAM, **-t:** écrit le temps pris par phase de recherche, **[contigs.fasta]:** fichiers de contigs, **-p:** nombre de threads d'alignement à lancer, **[reads_file.fasta]:** fichier contenant toutes les lectures (par lignes alternées pour les paires).
  - Pour Sspace :
    Pour ce scaffolder, les commandes de bowtie sont dans le code. Les paramètres pas défaut sont utilisés sauf **-T 1:** correspond à -p/–threads 1 de bowtie : nombre de threads d'alignement à lancer, **-g 0:** correspond à -v 0 de bowtie: pas de gaps autorisés pendant le mapping.

- **Sopra (Sopra_with_prebuilt_contigs/) :**
  perl s_prep_contigAseq_v1.4.6.pl -contig [contigs.fasta] -mate [reads_file.fasta] -a [output_folder]
  avec : **-contig:** fichier de contigs, **-mate:** fichier contenant toutes les lectures (par lignes alternées pour les paires), **-a:** dossier de sortie.

  Generation of the bowtie.sam file (see bowtie section)
  perl s_parse_sam_v1.4.6.pl -sam [bowtie.sam] -a [output_folder]
  avec : **-sam:** fichier d'alignement généré par bowtie au format sam, **-a:** dossier de sortie (même qu'avant).

  perl s_read_parsed_sam_v1.4.6.pl -parsed [bowtie.sam_parsed] -d 300 -a [output_folder]
  avec : **-parsed:** fichier d'alignement généré par bowtie et parsé à l'étape précédente, **-d:** taille d'insert, **-a:** dossier de sortie (même qu'avant).

  perl s_scaf_v1.4.6.pl -w 2 -o [orientdistinfo_c5] -a [output_folder]
  avec : **-w:** nombre minimum de liens de paire entre 2 contigs, **-o:** fichier orientdistinfo généré à l'étape précédente, **-a:** dossier de sortie (même qu'avant).

- **Opera :**
  bin/opera [contigs.fasta] [bowtie.bam] [output_folder]
  avec : **[contigs.fasta]:** fichier de contigs, **[bowtie.bam]:** fichier d'alignement généré par bowtie au format bam, **[output_folder]:** dossier de sortie.

- **Sspace :**
  perl Sspace_Basic_v2.0.pl -l [libraries.txt] -s [contigs.fa] -k 4 -z 150 -x 1 -b [scaffold_x_1]

avec : **[libraries.txt]:** fichier librairie contenant les chemins vers les librairies de lectures avec la taille d'insert, l'erreur et l'indication mate-pair ou paired-end, **[contigs.fa]:** fichier fasta contenant les séquences de contigs utilisées pour l'extension. Les paires insérées sont mappées aux contigs étendus et non-étendus, **-k:** nombre minimum de liens (paires de lectures) pour calculer les scaffolds, **-z:** longueur minimum des contigs pour le scaffolding. Filtre les contigs qui sont plus petits que -z, **-x:** indique si on étend les contigs de -s en utilisant les paires de lectures de -l, **-b:** nom de base pour les fichiers de sortie.

- **Quast :**
  python quast.py -o [output_folder] -R [reference_genome.fasta] –gene-finding (–eukaryote) (–scaffolds) [query_file.fasta]
  avec : **-o:** dossier où enregistrer tous les fichiers résultats, **-R:** fichier du génome de référence, **–gene-finding:** prédire les gènes avec GeneMark.hmm pour les prokaryotes (par défaut), GlimmerHMM pour les eukaryotes (–eukaryote), **–eukaryote:** le génome est eukaryote (prokaryote par défaut), **–scaffolds:** les assemblages sont des scaffolds, les couper et ajouter les contigs dans la comparaison (par défaut non-scaffold).

expe/res

E Sopra : Parsing et prétraitement

Reprenons le détail du processus de Sopra en commençons par les fichiers lançant des calculs de préprocessing :

1. **perl s_prep_contigAseq_v1.4.6.pl -contig [contigs.fasta] -mate [reads_file.fasta] -a [output_folder]**
   - Entrée :
     - Fichier de contigs :
       ">contig_name1 1 length
       ACGT... (sur plusieurs lignes)
       >contig_name2 1 length
       CAAT... (idem)"
     - Librairie :
       ">read_name1/1
       ACGT...
       >read_name1/2
       CAAT...
       >read_name2/1"
   - Sortie :
     - contigs_sopra.fasta :
       ">contig_name1 1 length
       ACGT... (sur 1 lignes)
       >..."

     reads _sopra.fasta :
       ">s1a
       ACGT...
       >s1b
       CAAT...
       >s2a"
     - structure tig_info stored in div/tig_info :
       tig_info->contig'length' : longueur du contig
       tig_info->contig'cov' : 0
       tig_info->1'seq_l_u' : 0
       tig_info->1'seq_l_d : 0
2. **Generation of the bowtie.sam file (see bowtie section)**
3. **perl s_parse_sam_v1.4.6.pl -sam [bowtie.sam] -a [output_folder]**
   - Entrée :

- sam_file :
  "@HD... (title) @SQ SN:name LN:length (contigs) @SQ... ... @PG ID:bowtie... CL:command read_name flag name start 255 cigar * 0 0 CGAT... quality"

- Sortie :

  - div/copynumber.txt :
    read1occurences read2occurences... (classés par ordre décroissant)
  - div/coverage_distribution.txt
    contig_number coverage (ou "" si longueur = 0)

sam_file _parsed :
  orientation(+/-) contig_number start_position nb_occurences longueur (seulement pour les paires de lectures)

  - structure tig_info stored in div/tig_info_cov :
    tig_info->contig'cov' : $\frac{somme(longueurdeslectures)}{Longueurducontig}$
    tig_info->1'seq_l_u' : longueur de la plus petite lecture
    tig_info->1'seq_l_d : longueur de la plus longue lecture

4. **perl s_read_parsed_sam_v1.4.6.pl -parsed [bowtie.sam_parsed] -d 300 -a [output_folder]**

- Entrée :

  - sam_file_parsed (voir au-dessus)

- Sortie :

  - div/read_length_[sam_file_parsed].txt :
    read1a_length read1b_length read2a_length... (si read_occurences < nombre maximum de copies d'une lecture)
  - div/insertsize_distribution_[sam_file_parsed]_[insert_size].txt :
    empirical_insert_size_pair_1... (idem et si contig_read1a = contig_read1b et si orientation_read1a != orientation_read1b)
  - structure ORI_DIS stored in orientdistinfo_c :
    ORI_DIS->contig1contig2'P' = nombre de paires →← ou ←→ entre contig1 et contig2
    ORI_DIS->contig1contig2'N' = nombre de paires →→ ou ←←← entre contig1 et contig2
    ORI_DIS->contig1contig2'PD' += orientation_read1*(-insert_size + read1_start - read2_start) si (←→), orientation_read1*(insert_size + read1_start - read2_start) si (→←)
    ORI_DIS->contig1contig2'ND' += insert_size + read1_start + read2_start si (→→), -insert_size + read1_start + read2_start si (←←)

Ensuite vient le script de scaffolding en lui-même : **perl s_scaf_v1.4.6.pl -w 2 -o [orientdistinfo_c5] -a [output_folder]**

References

[1] Rémi Coletta Simon de Givry Philippe Leleux Nicolas Briot, Annie Chateau and Thomas Schiex. An integer linear programming approach for genome scaffolding. In *WCB2014*, 2014.

[2] Aurélie Favier. *Décompositions fonctionnelles et structurelles dans les modèles graphiques probabilistes appliquées à la reconstruction d'haplotypes.* PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2011.

[3] Simon de Givry. Rapport d'habilitationa diriger des recherches. 2011.

[4] Rayan Chikhi. *Computational methods for de novo assembly of next-generation genome sequencing data.* PhD thesis, Johannes Gutenberg-Universität Mainz, 2012.

[5] Jared T Simpson and Richard Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome research*, 22(3):549–556, 2012.

[6] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.

[7] Rayan Chikhi, Guillaume Rizk, et al. Space-efficient and exact de bruijn graph representation based on a bloom filter. In *WABI*, pages 236–248, 2012.

[8] Marten Boetzer, Christiaan V Henkel, Hans J Jansen, Derek Butler, and Walter Pirovano. Scaffolding pre-assembled contigs using sspace. *Bioinformatics*, 27(4):578–579, 2011.

[9] Song Gao, Wing-Kin Sung, and Niranjan Nagarajan. Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *Journal of Computational Biology*, 18(11):1681–1691, 2011.

[10] Nilgun Donmez and Michael Brudno. Scarpa: scaffolding reads with practical algorithms. *Bioinformatics*, 29(4):428–434, 2013.

[11] Adel Dayarian, Todd P Michael, and Anirvan M Sengupta. Sopra: Scaffolding algorithm for paired reads via statistical optimization. *BMC bioinformatics*, 11(1):345, 2010.

[12] Ben Langmead, Cole Trapnell, Mihai Pop, Steven L Salzberg, et al. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[13] René L Warren, Granger G Sutton, Steven JM Jones, and Robert A Holt. Assembling millions of short dna sequences using ssake. *Bioinformatics*, 23(4):500–501, 2007.

[14] Leena Salmela, Veli Mäkinen, Niko Välimäki, Johannes Ylinen, and Esko Ukkonen. Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, 27(23):3259–3265, 2011.

[15] Dent Earl, Keith Bradnam, John St John, Aaron Darling, Dawei Lin, Joseph Fass, Hung On Ken Yu, Vince Buffalo, Daniel R Zerbino, Mark Diekhans, et al. Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Genome research*, 21(12):2224–2241, 2011.

[16] Keith R Bradnam, Joseph N Fass, Anton Alexandrov, Paul Baranay, Michael Bechner, Inanç Birol, Sébastien Boisvert, Jarrod A Chapman, Guillaume Chapuis, Rayan Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2(1):1–31, 2013.

[17] Martin Hunt, Taisei Kikuchi, Mandy Sanders, Chris Newbold, Matthew Berriman, and Thomas D Otto. Reapr: a universal tool for genome assembly evaluation. *Genome Biol*, 14(5):R47, 2013.

[18] Martin Hunt, Chris Newbold, Matthew Berriman, and Thomas D Otto. A comprehensive evaluation of assembly scaffolding tools. *Genome biology*, 15(3):R42, 2014.

[19] Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

[20] Steven L Salzberg, Adam M Phillippy, Aleksey Zimin, Daniela Puiu, Tanja Magoc, Sergey Koren, Todd J Treangen, Michael C Schatz, Arthur L Delcher, Michael Roberts, et al. Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567, 2012.

[21] Tanja Magoc, Stephan Pabinger, Stefan Canzar, Xinyue Liu, Qi Su, Daniela Puiu, Luke J Tallon, and Steven L Salzberg. Gage-b: an evaluation of genome assemblers for bacterial organisms. *Bioinformatics*, 29(14):1718–1725, 2013.

[22] Pramila Nuwantha Ariyaratne and Wing-Kin Sung. Pe-assembler: de novo assembler using short paired-end reads. *Bioinformatics*, 27(2):167–174, 2011.

[23] Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones, and Inanç Birol. Abyss: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123, 2009.

[24] Jarrod A Chapman, Isaac Ho, Sirisha Sunkara, Shujun Luo, Gary P Schroth, and Daniel S Rokhsar. Meraculous: de novo genome assembly with short paired-end reads. *PloS one*, 6(8):e23501, 2011.

[25] Rayan Chikhi, Guillaume Chapuis, Dominique Lavenier, et al. Parallel and memory-efficient reads indexing for genome assembly. In *Parallel Bio-Computing 2011*, 2011.

[26] Pavel A Pevzner, Haixu Tang, and Michael S Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[27] Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, Shengting Li, Gao Shan, Karsten Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20(2):265–272, 2010.

[28] Sante Gnerre, Iain MacCallum, Dariusz Przybylski, Filipe J Ribeiro, Joshua N Burton, Bruce J Walker, Ted Sharpe, Giles Hall, Terrance P Shea, Sean Sykes, et al. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, 108(4):1513–1518, 2011.