Faculté des Sciences de Montpellier

–

Institut des Sciences de l'Évolution de Montpellier

Visualisation de comparaisons d'assemblages de génomes

Samia Benarbia

Encadrante de stage : Madame Sèverine Bérard

7 juin - 23 juillet 2021

Remerciements

Avant toute chose, je tiens à exprimer toute ma reconnaissance à mon encadrante de stage Madame Sèverine Bérard pour m'avoir permis de faire un premier pas dans le monde de la bioinformatique. Je la remercie de m'avoir encadrée, orientée, aidée et pour ses conseils précieux qui m'accompagneront à l'avenir.

Je tiens également à remercier l'Université de Montpellier ainsi que l'Institut des Sciences de l'Évolution de Montpellier (ISEM) pour m'avoir accueillie dans leur structure, me permettant d'effectuer ce stage.

Table des matières

Table des figures

1 Introduction

Dans le cadre de ma deuxième année de licence informatique à la Faculté des Sciences de Montpellier, j'ai choisi de réaliser un stage d'été à l'Institut des Sciences de l'Évolution de Montpellier (ISEM). Il s'agit d'un laboratoire de recherche traitant des problématiques liées à la biodiversité comme ses mécanismes d'évolution, son origine et ses dynamiques. Les recherches sont menées sur des biodiversités passées ou actuelles.

Étant sensible au domaine de la bioinformatique et envisageant d'effectuer un master dans ce domaine après ma licence, il m'a paru pertinent d'effectuer ce stage afin d'avoir une première approche dans un des champs de ce domaine. Ayant eu Madame Bérard comme enseignante lors de ma première année de licence et m'étant intéressée à ses travaux dans l'étude des mécanismes évolutifs, j'ai pris contact avec elle afin de lui exposer mon projet de stage. Elle a accepté d'être mon encadrante et nous avons discuté ensemble de mes objectifs ainsi que de mes attentes vis à vis de ce stage.

S'agissant de mon premier stage en bioinformatique, nous avons choisi un sujet qui me permettrait de toucher à de multiples facettes de la discipline. Nous nous sommes orientées sur un sujet traitant des assemblages de génomes et plus particulièrement nous nous sommes fixées comme objectif de proposer une visualisation, sous forme de graphe, d'une comparaison d'assemblages de génomes. Ce travail permettra à une doctorante de visualiser les résultats de son algorithme de comparaison.

Pour cela nous parlons dans un premier temps du passage de l'aspect biologique, à partir de l'ADN, à l'aspect informatique. L'ADN étant composé de quatres bazes azotées, adénine, cytosine, guanine et thymine (A, C, G, T), le séquençage nous permet de connaître l'enchaînement de ces nucléotides afin de cartographier des génomes. Ce séquençage retranscrit l'enchaînement des bases azotées sous forme d'un texte (avec les lettres A, C, G, T et N si la base azotée n'est pas connue). Ces séquences de fragments sont appellées des *reads* qui sont ensuite analysées informatiquement pour pouvoir effectuer les assemblages des génomes. Nous cherchons ensuite, dans le cadre de ce stage, à produire une visualisation de ces assemblages afin de nous permettre de constater visuellement les zones de chevauchements, d'inversions et de correspondances entre des scaffolds et une référence. L'objectif plus général dans le cadre du doctorat, étant de réduire le nombre de scaffolds pour représenter le plus précisément possible le génome de l'organisme étudié.

Pour cela nous commençons par une partie bibliographique afin d'expliquer plus en détail le passage de la biologie au traitement informatique des assemblages de génomes. Nous verrons ensuite comment obtenir un visuel d'une comparaison d'assemblage en parlant du format de fichier DOT ainsi que du logiciel Graphviz. Enfin, dans un dernier temps, nous verrons comment automatiser l'écriture du fichier DOT afin d'obtenir une visualisation de comparaison d'assemblage à partir d'un fichier texte.

2 Bibliographie

2.1 Génomes

Le mot *"génome"* est composé des mots *"gène"* et *"chromosome"* [5]. Le **génome** contient l'ensemble de l'information génétique d'un organisme. Ces informations sont stockées dans l'**ADN** (Acide DésoxyriboNucléique) pour l'ensemble des êtres vivants, exceptés les virus pour lesquels l'information est stockée dans l'**ARN** (Acide RiboNucléique). Cet ensemble d'information est représenté sur les **chromosomes** par une suite linéaire de **gènes** [16].

Il existe une distinction entre les **procaryotes** et les **eucaryotes**. Chez les procaryotes, le noyau cellulaire est mêlé au cytoplasme, le génome est donc principalement composé d'un unique chromosome circulaire. Tandis que chez les eucaryotes, les cellules possèdent un noyau et des organites délimités par des membranes [15] (voir la figure 1).

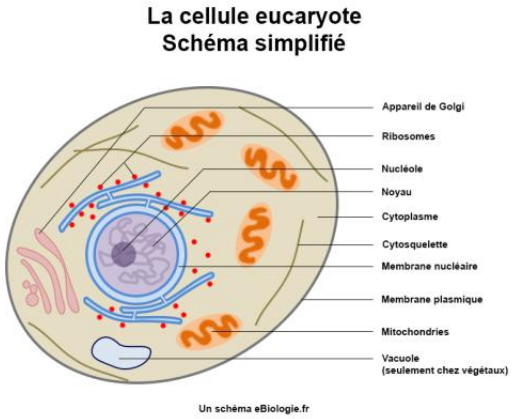

FIGURE 1 – Schéma simplifié d'une cellule eucaryote [14]

L'ADN est composé de quatre **bases azotées** ou **nucléotides** : l'adénine, la cytosine, la guanine et la thymine (A, C, G, T). Ce sont ces nucléotides qui constituent le code génétique [19] (voir la figure 2 ci-dessous).

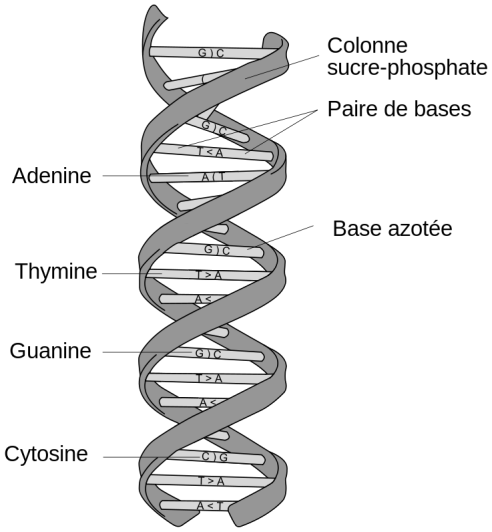

FIGURE 2 – Composition d'un Acide DésoxyriboNucléique [19]

2.2 Techniques de séquençage de génomes [12]

Le séquençage de l'ADN permet de connaître l'enchaînement des nucléotides (A, C, G, T) et de cartographier les génomes, ce qui permet de connaître les informations biologiques d'un organisme. Il existe actuellement plusieurs banques de données publiques des génomes de plus d'un millier d'espèces, ce qui permet l'étude de l'évolution des espèces à partir de leur matériel génétique complet comme par exemple la base de donnée *GenBank* [18]. Il y a une utilisation quasi exclusive des **séquenceurs automatiques** ce qui permet une plus haute précision du séquençage [3].

Le séquençage des génomes permet de retranscire les gènes sous forme d'un texte (grâce aux nucléotides A, C, G, T et N si la nucléotide n'est pas connue) pouvant être ensuite traité par les bioinformaticiens.

2.2.1 Technique Maxam-Gilbert

Cette technique créée en 1977 est aujourd'hui quasiment abandonnée. Il s'agit d'une méthode chimique de séquençage. Elle repose sur la propriété qu'ont certains agents chimiques comme l'acide formique (CH_2O_2) ou encore le diméthylsulfate (*DMS*) sur les bases de l'ADN. Ces réactifs vont venir cliver [1] après chacune des bases A, C, G, T. De la pipéridine [2] est ensuite ajoutée ce qui vient casser les brins d'ADN au niveau des bases précédemment modifiées. L'ADN à séquencer est marqué à une extrémité par un marqueur radioactif (souvent l'isotope phosphore 32) puis il est lu par autoradiographie après traitement chimique des fragments (voir la figure 3 ci-dessous).

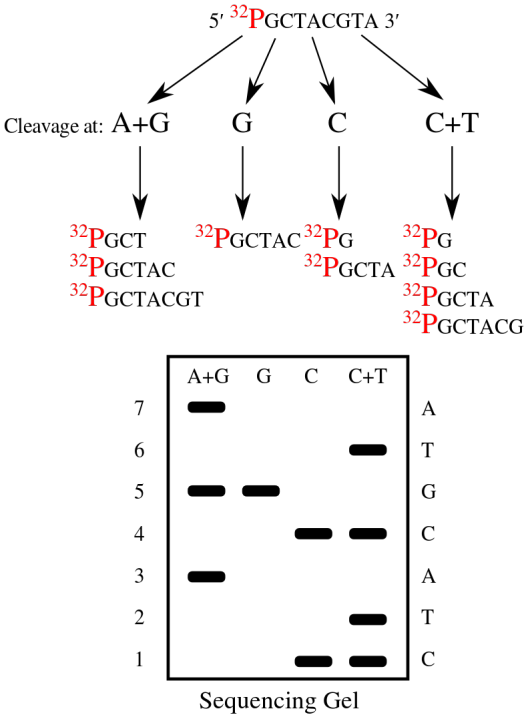

FIGURE 3 – Méthode de séquençage de Maxam et Gilbert [24]

1. il s'agit d'un processus de décomposition de grosses molécules en scindant leurs liaisons intramoléculaires[20]
2. la $C_5H_{11}N$ a des propriétés de base faible et est souvent utilisée comme solvant en synthèse organique [25]

2.2.2 Technique de Sanger

La technique de Sanger a largement dépassé la méthode Maxam-Gilbert. Le principe de cette technique est de tout d'abord amplifier l'ADN cible par **PCR** (Polymerase Chain Reaction ou polymérisation en chaîne). Cela permet d'obtenir un grand nombre de copies identiques d'un fragment d'ADN de l'ordre du milliard. Elle fonctionne en trois étapes, en programmant des cycles consécutifs de montée et de baisse de température : la **dénaturation** (les deux brins d'ADN sont séparés par chauffage), l'**hybridation** (lors de la baisse de température, des amorces constituées de courts fragments d'ADN s'hybrident sur les brins d'ADN) et enfin l'**élongation** (la Taq polymérase, un ADN polymérase, complète la synthèse du brin d'ADN à partir de l'amorce grâce aux oligonucléotides présents dans le milieu de réaction), voir la figure 4 ci-dessous [7].

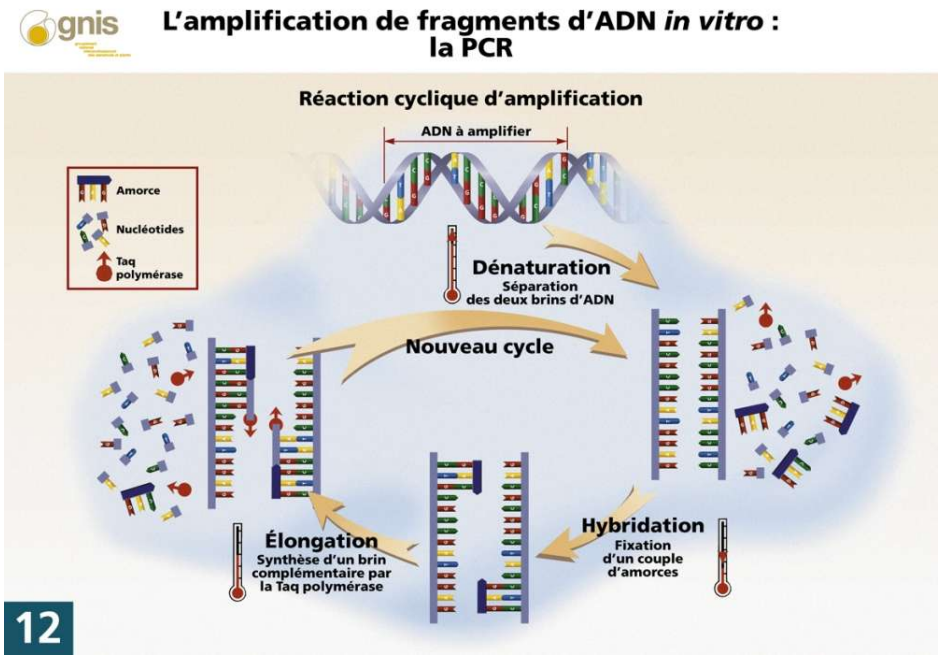

FIGURE 4 – Schéma représentant les étapes d'une PCR [7]

Après cette première étape de PCR, il faut ensuite dénaturer l'ADN cible afin d'obtenir un ADN simple brin. Ensuite c'est une ADN polymérase qui va effectuer la synthèse de l'ADN complémentaire à partir d'une amorce définie au préalable (elle peut être identique ou différente de celle utilisée pour la PCR). Les deux fragments n'auront pas nécessairement la même longueur. En effet, l'ADN polymérase va ajouter aléatoirement des désoxyribonucléotides-triphosphates (dNTP) complémentaires et des didéoxyribonucléotides triphosphates (ddNTP) et lorsqu'un ddNTP est incorporé à la place d'un dNTP, l'ADN polymérase ne peut plus continuer sa polymérisation. Statistiquement, au cours de la réaction, pour chaque « base » de l'ADN cible, au moins une fois, un ddNTP complémentaire sera incorporé à la place d'un dNTP.

S'en suit une étape d'analyse de la réaction. Il existe différentes méthodes mais celle la plus utilisée aujourd'hui est **l'électrophorèse capillaire** réalisée sur un **automate de séquençage**. Chaque fragment qui contient un ddNTP marqué par un fluorophore est excité par un laser lors de sa migration. On obtient ainsi un signal analysé. On peut ensuite réaliser une **analyse informatique** des signaux ce qui permet d'obtenir la séquence étudiée sous la forme d'un **électrophorégramme** (voir la figure 5 ci-sessous). On peut ensuite utiliser des logiciels d'analyse des séquences.

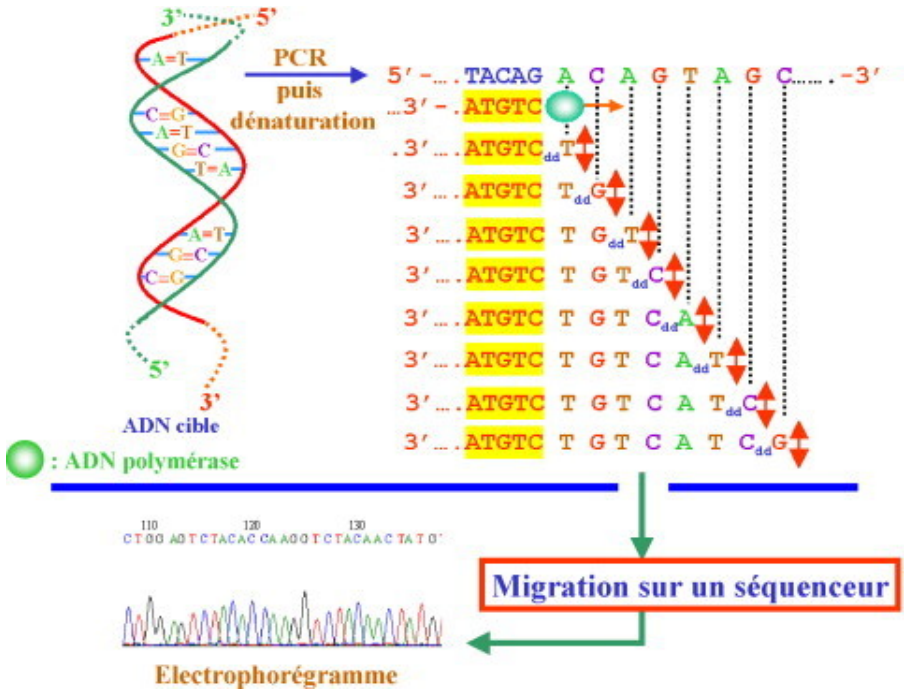

FIGURE 5 – Technique de Sanger [12]

2.2.3 Nouvelles technologies de séquençage et stockage des données

Depuis l'apparition des nouvelles technologies de séquençage (Next-Generation Sequencing ou NGS) dans les années 2000, il faut savoir gérer environ 15 000 fois plus de données par jour qu'un séquenceur Sanger (voir la figure 6). En raison des progrès et mises à niveau de la mémoire des disques dur et de la RAM, les séquenceurs peuvent stocker les téraoctets d'informations générées par le séquençage du génome.

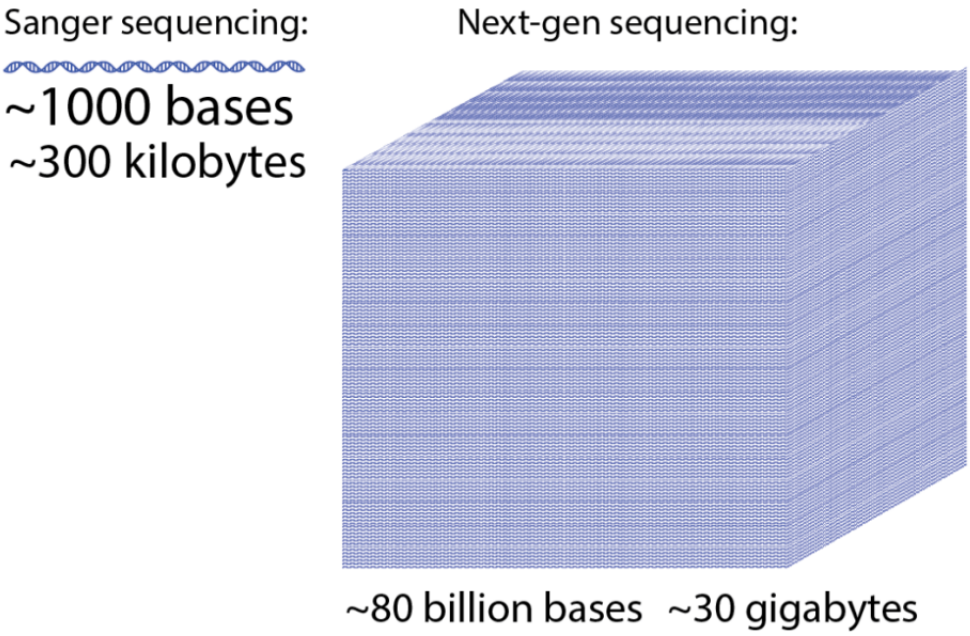

FIGURE 6 – Séquençage Sanger comparé au NGS [4]

Afin de pouvoir interpréter les résultats des séquençages, il est entre autres possible d'utiliser des algorithmes de correspondance de motifs, des modèles mathématiques, le traitement d'image, etc. Il également intéressant de réaliser des simulations afin de prédire comment certains systèmes biologiques peuvent réagir dans différents environnements.

À la fin du séquençage, les séquences des fragments, appellées *reads* sont sauvegardées dans un fichier FASTQ [22] (format de fichier permettant de stocker des séquences biologiques et les scores de qualité associé). Ces fichiers textes seront ensuite traités et analysés informatiquement afin de réaliser les assemblages des génomes.

2.3 Assemblages de génomes

Après le séquençage des génomes, l'étape de l'assemblage consiste à reconstruire une séquence initiale du génome à partir des **reads** obtenus par le séquençage de cette même séquence [13].

Il existe plusieurs techniques d'assemblage, notamment depuis l'apparitions des NGS (nouvelles technologies de séquençage) [11].

2.3.1 Assemblage *de novo* [6]

L'algorithme prend en entrée des fragments d'ADN et renvoie en sortie des longues séquences de **régions contigues** (séquences génomiques continues et ordonnées générées par l'assemblage des clones d'une bibliothèque génomique) déterminées selon les **chevauchements** entre les fragments. Puis on assemble les **contigs** [21] en **scaffolds** (une série de contigs qui sont dans le bon ordre mais pas nécessairement connectés dans un tronçon continu de séquence) qui sont ensuite assemblés pour former la **séquence finale** (voir la figure 7 ci-dessous).

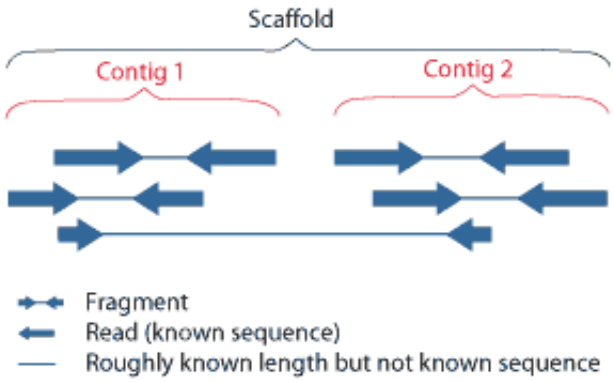

FIGURE 7 – Étapes du scaffolding [26]

2.3.2 Assemblage *overlap-layout-consensus* [6]

Cette méhode d'assemblage est consitutée de trois étapes :
- *Overlap* : qui consiste à déterminer les **chevauchements** parmi les **séquences shotgun** (voir cet article à propos de la technique *shotgun*)
- *Layout* : afin de déterminer **l'ordre** des séquences shotgun
- *Consensus* : pour déterminer la **séquence des contigs**. Le layout donne la position approximative de chaque fragment puis il faut trouver la séquence de consensus, à partir d'un **alignement multiple**.

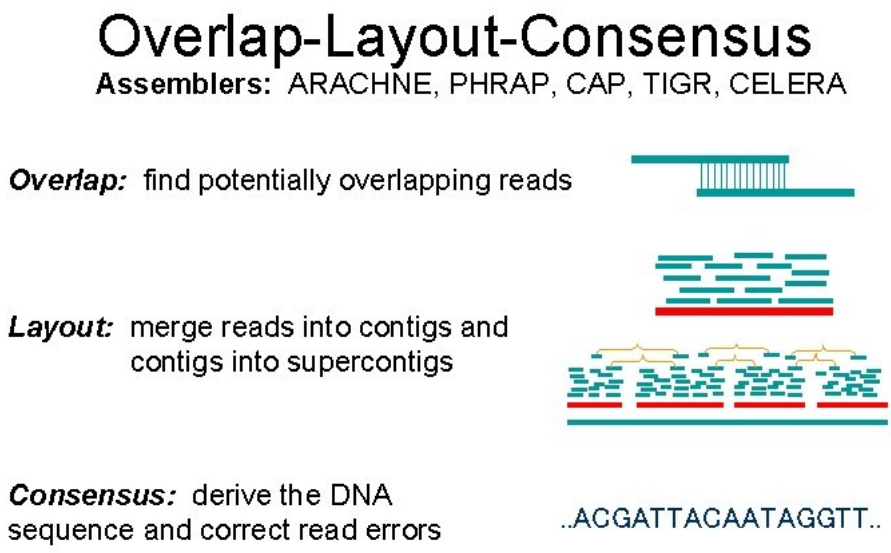

FIGURE 8 – Schéma simplifié d'un assemblage overlap-layout-consensus [17]

3 Visualisation des séquences d'ADN

Un génome est composé d'un nombre de chromosomes de l'ordre de 10 à 100. Cependant après le séquençage et l'assemblage d'un génome, on obtient plutôt un nombre de scaffolds de l'ordre du millier. Une partie du travail d'Elisa Henrion-Gueneau, une doctorante encadrée par Madame Bérard, consiste à réduire ce nombre afin d'obtenir le génome complet de l'organisme étudié en comparant deux à deux un scaffold et une référence.

Nous allons donc chercher à produire une visualisation de ces assemblages de génomes composés de scaffolds ainsi que de la référence. Pour cela nous utilisons un fichier texte composé de nombres. Ces nombres représentent les correspondances entre le scaffold et la réference à partir du read obtenu par l'assemblage (voir figure 9 ci-dessous).

```
1       0       750     0       750
1       625     3050    621     3046
1       2300    4375    2696    4771
1       3625    5425    5400    7200
1       5300    6675    7077    8452
1       5900    29450   8433    31983
1       29000   44000   32345   47345
1       43900   45075   47262   48437
1       44925   45875   48286   49236
2       0       1675    49244   50919
2       1175    22675   51033   72533
2       21850   22825   72212   73187
2       22675   23675   73038   74038
2       22925   45425   74023   96523
```

Figure 9 – Affichage simplifié des correspondances entre les scaffolds et la référence, qui sera utilisé pour produire la visualisation. En lisant ligne par ligne : le premier chiffre correspond au numéro de scaffold, le deuxième au début de segment du scaffold, le troisième à la fin de segment du scaffold, le quatrième au début de segment de la référence et le cinquième à la fin de segment de la référence.

3.1 Visualisation avec Excalidraw

Au début de mon travail sur la visualisation, j'ai été amenée à utiliser l'application open source **Excalidraw** [1] afin d'avoir un premier aperçu de comment est-ce que nous pouvons représenter les assemblages et les comparaisons entre les scaffolds et la référence.

J'ai choisi de réaliser une représentation verticale avec une alternance de quatre couleurs de rectangles (voir la figure 10 ci-dessous). Chaque rectangle sur le scaffold ou la référence représente une zone de correspondance reliée à son homologue par un segment de même couleur.

Pour cette première représentation, j'ai utilisé le fichier simplifié comprenant seulement les deux premiers scaffolds (figure 9).

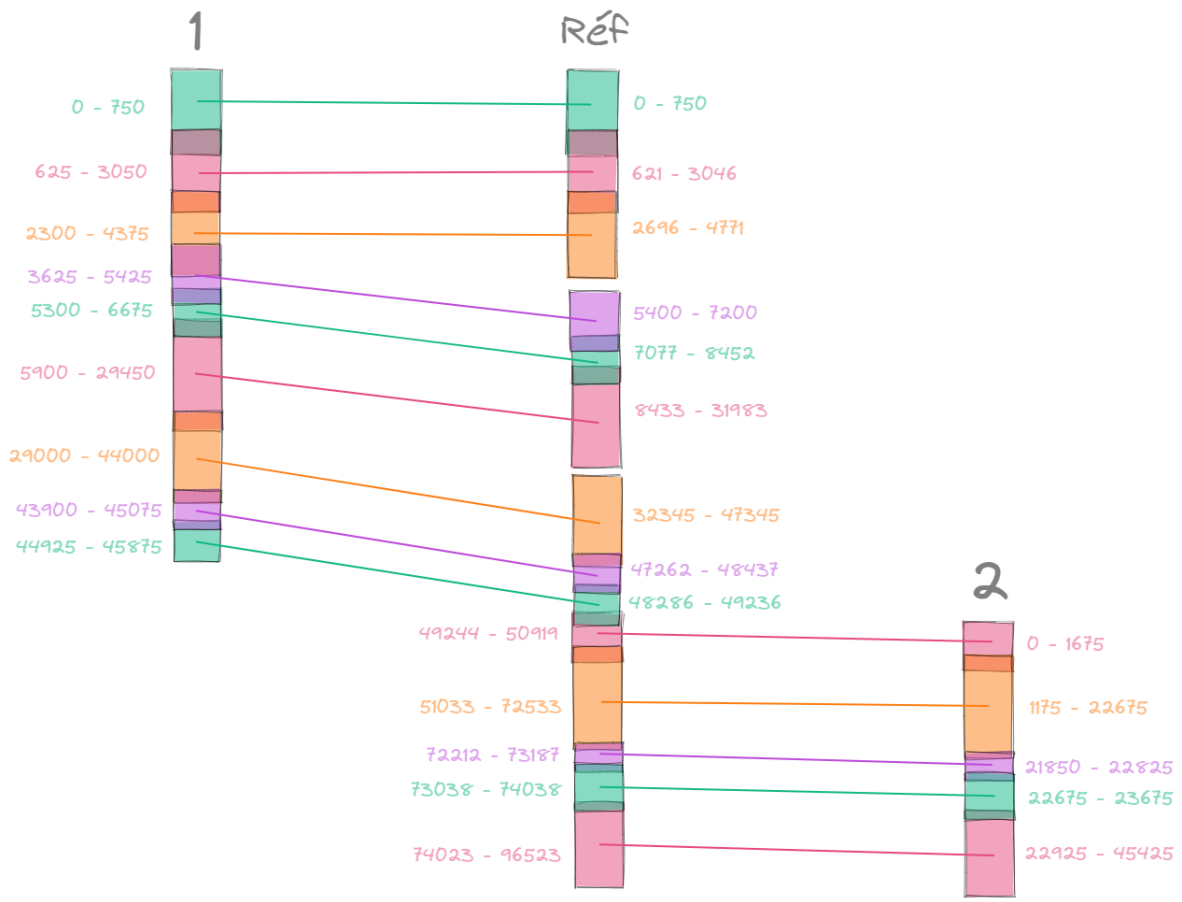

FIGURE 10 – Première représentation de la comparaison d'assamblage avec l'application Excalidraw

Cette méthode de visualisation permet d'avoir une première idée du résultat mais il s'agit seulement d'un aperçu car avec les possibilités de modélisation d'Excalidraw nous sommes assez limités. Par exemple, nous ne pouvons pas entrer en dur les dimensions des rectangles, nous pouvont seulement leur donner une taille approximative à la souris, nous ne respectons donc pas les proportions. De plus, il n'y a aucun moyen d'automatisation avec cette méthode.

3.2 Le langage *DOT* [9]

Le langage DOT est un langage permettant d'encoder des graphes dans un format texte. Il fait partie de la suite de logiciels Graphviz qui sont open source [23].

On peut utiliser la ligne de commande suivante afin de générer un fichier dot : *"dot -T**ext** fichiersource .dot -o fichierdestination.**ext**"*, **ext** étant l'extension associée au format voulu (comme par exemple **pdf**).

La commande *graph* permet de créer un **graphe non orienté** tandis que la commande *digraph* permet de générer un **graphe orienté**. On peut également créer des **sous-graphes** avec la commande *subgraph*. On peut créer au sein de nos graphes des **nœuds de départ et de fin** à l'aide des commandes *start* et *end* qui sont les noms de ces deux nœuds spéciaux.

Afin de spécifier des caractéristiques à un nœud en général (les nœuds *start* et *end* compris), il suffit de les écrire entre crochet tel que : *node[caractéristiques]*. Comme caractéristiques il y a par exemple *label* qui permet de donner un nom à n'importe quel objet du fichier dot (un graphe, un sous graphe, un segment, un nœud, etc). Il existe aussi des commandes afin de gérer la taille d'un graphe (*size*), la forme ou la couleur d'un nœud (*shape*, *color* [8]) ou encore la commande *style* qui

permet de spécifier si l'on souhaite l'écriture en gras, si l'on souhaite remplir ou hachurer un nœud ou un graphe.

Dans notre problématique de représentation d'assemblage d'ADN et de visualisation des scaffolds ainsi que la visualisation des paires, toutes les commandes précédemment évoquées nous permettent de paramétrer notre graphe et en particulier nos nœuds afin de leur donner la forme et la taille nécessaires.

Une autre commande va nous intéresser : la commande *pos*, permettant de fixer les coordonnées d'un point dans le graphe. Cela va nous permettre de fixer chaque milieu de nœud au bon endroit dans le graphe et de manière précise afin de pouvoir construire le scaffold.

Voici quelques exemples de graphes que j'ai pu créer, comme entraînement dans un premier temps, puis spécifiques à notre problématique :

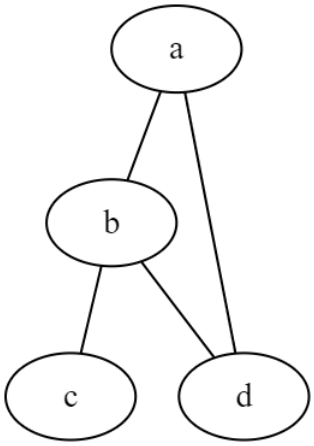

Figure 11 – Exemple de graphe non orienté

```
graph mon_graphe_no {
    a -- b -- c;
    b--d;
    a--d;
}
```

Figure 12 – Code associé au graphe non orienté

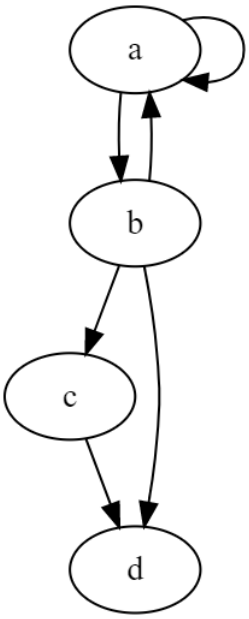

Figure 13 – Exemple de graphe orienté

```
digraph mon_graphe_o {
    a -> a;
    a -> b -> a;
    b -> c -> d;
    b -> d;
}
```

Figure 14 – Code associé au graphe orienté

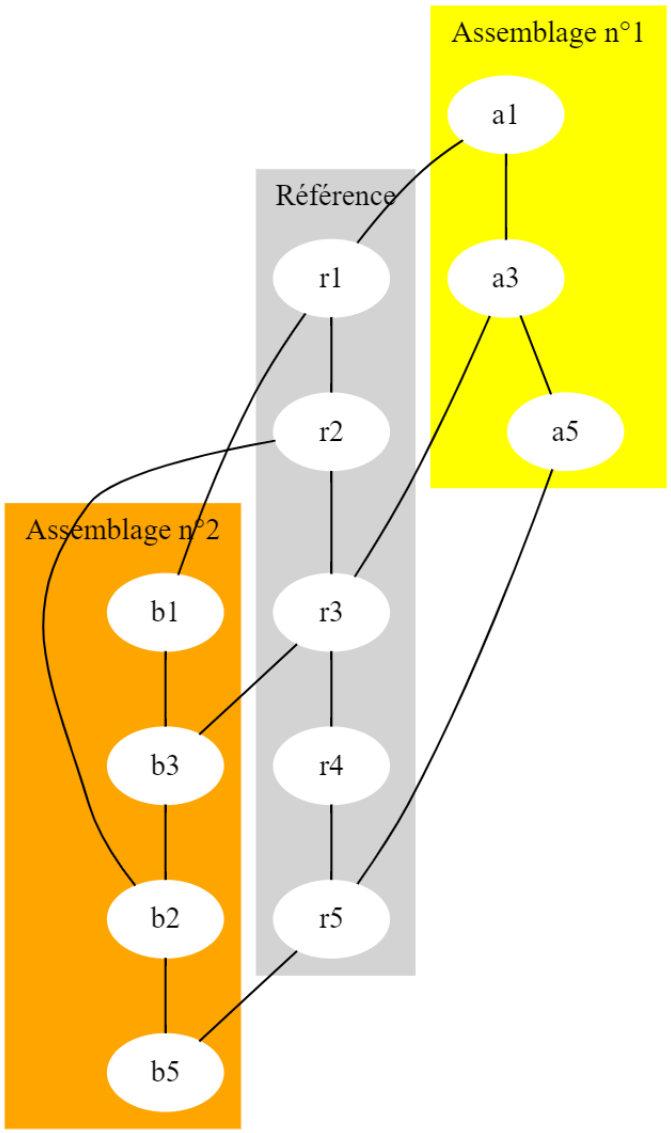

FIGURE 15 – Premier essai de représentation de scaffolds issu du code de la figure 16

On peut voir sur la figure 15 que le placement des nœuds est aléatoire et qu'il est nécessaire de fixer en dur la position des nœuds afin d'obtenir une visualisation cohérente.

```
graph parallelismes {

    subgraph cluster_0 {
        label="Assemblage n°1";
        style=filled;
        color=yellow;
        node [style=filled,color=white];
        a1 -- a3 -- a5;
    }

    subgraph cluster_1 {
        label="Référence";
        style=filled;
        color=lightgrey;
        node [style=filled,color=white];
        r1 -- r2 -- r3 -- r4 -- r5;
    }

    subgraph cluster_2 {
        label="Assemblage n°2";
        style=filled;
        color=orange;
        node [style=filled,color=white];
        b1 -- b3 -- b2 -- b5;
    }

    a1 -- r1 -- b1;
    r2 -- b2;
    a3 -- r3 -- b3;
    a5 -- r5 -- b5;

}
```

FIGURE 16 – Code associé au premier essai de représentation de scaffolds (graphe en figure 15)

Après avoir pris en main le langage dot, j'ai pu créer le graphe correspondant au fichier sur lequel nous nous basons (voir la figure 9).

Dans un premier temps, j'ai réalisé le graphe "à la main", c'est-à-dire que j'ai entré les caractéristiques et les positions des nœuds un à un afin de construire la représentation finale (voir les figures 17, 18 et 19 ci-dessous).

```
graph parallelismes2 {

    layout=neato;

    node [shape=box, style=filled];

    a [height=45.875, width=1.5, pos="0,-22.5625!", style=empty, color=gray55];

    a0 [height=0.75, pos="-0.1,0!", color=bisque];
    a1 [height=2.425, pos="0.1,-1.4625!", color=darkseagreen1];
    a2 [height=2.075, pos="-0.1,-2.9625!", color=cadetblue2];
    a3 [height=1.8, pos="0.1,-4.15!", color=bisque];
    a4 [height=1.375, pos="-0.1,-5.6125!", color=darkseagreen1];
    a5 [height=23.55, pos="0.1,-17.3!", color=cadetblue2];
    a6 [height=15, pos="-0.1,-36.125!", color=bisque];
    a7 [height=1.175, pos="0.1,-44.1125!", color=darkseagreen1];
    a8 [height=0.95, pos="-0.1,-45.025!", color=cadetblue2];


    r [height=96.523, width=1.5, pos="8,-47.8935!", style=empty, color=gray55];

    r0 [height=0.75, pos="7.9,0!", color=bisque];
    r1 [height=2.425, pos="8.1,-1.4585!", color=darkseagreen1];
    r2 [height=2.075, pos="7.9,-3.3585!", color=cadetblue2];
    r3 [height=1.8, pos="8.1,-5.925!", color=bisque];
    r4 [height=1.375, pos="7.9,-7.3895!", color=darkseagreen1];
    r5 [height=23.55, pos="8.1,-19.833!", color=cadetblue2];
    r6 [height=15, pos="7.9,-39.47!", color=bisque];
    r7 [height=1.175, pos="8.1,-47.4745!", color=darkseagreen1];
    r8 [height=0.95, pos="7.9,-48.386!", color=cadetblue2];

    r9 [height=1.675, pos="8.1,-49.7065!", color=bisque];
    r10 [height=21.5, pos="7.9,-61.408!", color=darkseagreen1];
    r11 [height=0.975, pos="8.1,-72.3245!", color=cadetblue2];
    r12 [height=1, pos="7.9,-73.163!", color=bisque];
    r13 [height=22.5, pos="8.1,-84.898!", color=darkseagreen1];

```

FIGURE 17 – Code générant le premier graphe - Partie 1/2

```

    b [height=45.425, width=1.5, pos="16,-71.581!", style=empty, color=gray55];

    b0 [height=1.675, pos="15.9,-49.7065!", color=bisque];
    b1 [height=21.5, pos="16.1,-60.794!", color=darkseagreen1];
    b2 [height=0.975, pos="15.9,-71.2065!", color=cadetblue2];
    b3 [height=1, pos="16.1,-72.044!", color=bisque];
    b4 [height=22.5, pos="15.9,-83.044!", color=darkseagreen1];


    a0 -- r0 [style=bold, color=bisque];
    a1 -- r1 [style=bold, color=darkseagreen1];
    a2 -- r2 [style=bold, color=cadetblue2];
    a3 -- r3 [style=bold, color=bisque];
    a4 -- r4 [style=bold, color=darkseagreen1];
    a5 -- r5 [style=bold, color=cadetblue2];
    a6 -- r6 [style=bold, color=bisque];
    a7 -- r7 [style=bold, color=darkseagreen1];
    a8 -- r8 [style=bold, color=cadetblue2];

    b0 -- r9 [style=bold, color=bisque];
    b1 -- r10 [style=bold, color=darkseagreen1];
    b2 -- r11 [style=bold, color=cadetblue2];
    b3 -- r12 [style=bold, color=bisque];
    b4 -- r13 [style=bold, color=darkseagreen1];
}
```

FIGURE 18 – Code générant le premier graphe - Partie 2/2

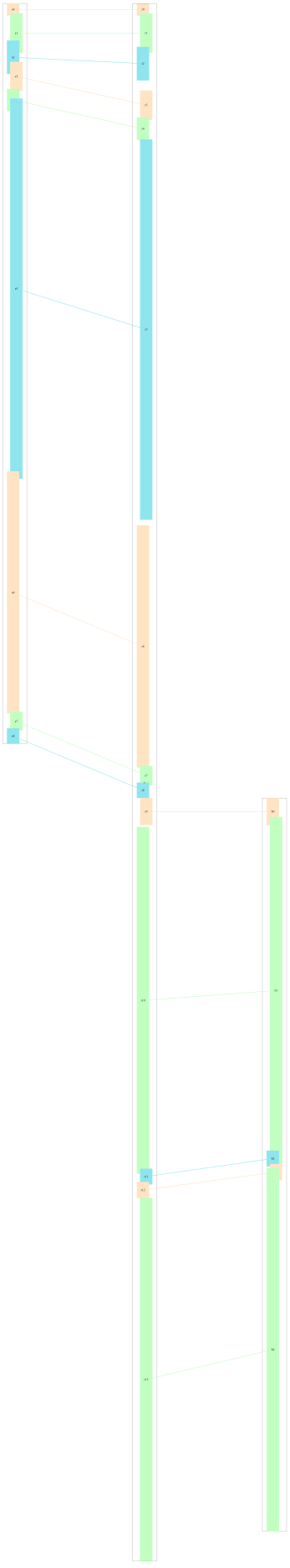

FIGURE 19 – Premier graphe représentant la référence ainsi que les deux premiers scaffolds

Dans un souci de lisibilité du graphe, nous avons fait le choix d'une alternance de trois couleurs et de décaler chaque nœud afin de bien observer les chevauchements. De plus, nous avons ajouté un grand nœud en arrière plan de la référence et de chacun des scaffolds afin de signifer qu'il s'agit bien d'un même bloc malgré des espaces qui peuvent exister entre certains nœuds.

3.3 Graphviz (*Graph Visualization Software*)

Graphviz est un ensemble d'outils open source qui permet de gérer des graphes définis à l'aide de scripts dans le langage DOT. C'est un logiciel libre créé par les laboratoires d'AT&T et distribué suivant l'Eclipse Public License.

J'ai choisi de travailler en ligne, sur le site `dreampuf.github.io/GraphvizOnline` [2] qui permet une visualisation simultanée du ou des graphes générés (voir la figure 20) ainsi que de télécharger le graphe sous différents formats d'image.

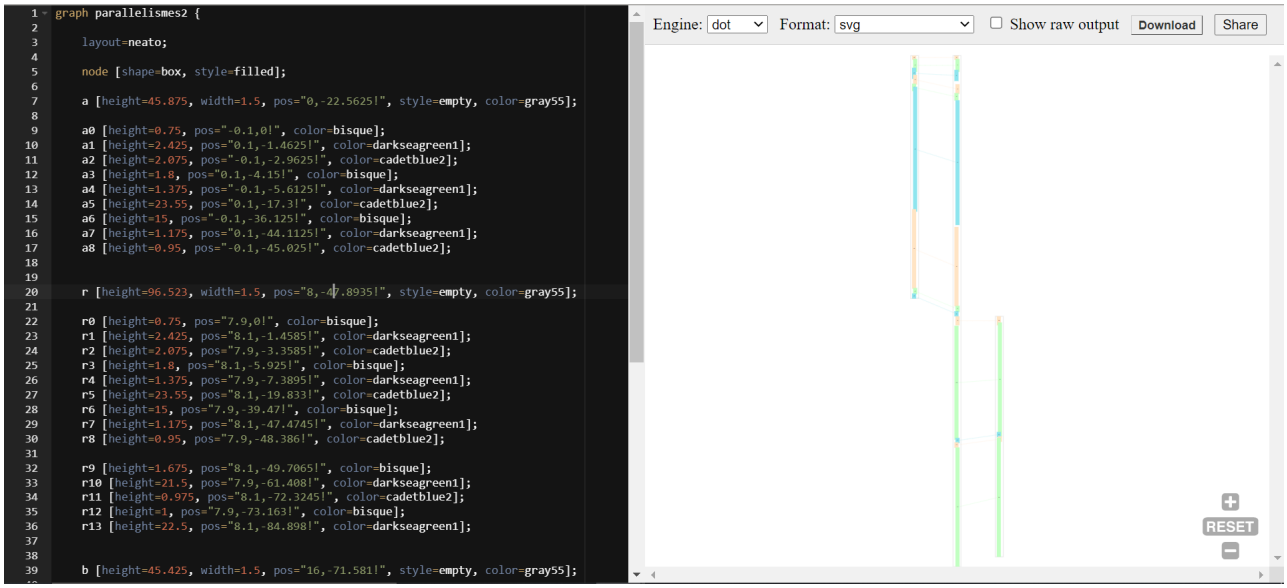

FIGURE 20 – Interface GraphvizOnline : la partie gauche est la console où l'on peut écrire le code tandis qu'à droite nous avons le visuel du graphe que le code génère en temps réel.

4 Automatisation de la visualisation : algorithmes [10]

Après avoir exploré comment créer le graphe à la main, nous passons à l'étape d'automatisation afin de prendre le fichier texte *résultat* en entrée et d'obtenir le fichier DOT permettant d'encoder le graphe de la comparaison d'assemblage en sortie.

Nous avons choisi d'écrire trois algorithmes dans un souci de clarté. L'algorithme 2 ***constr Noeud*** permet de construire un nœud et renvoie son numéro, l'algorithme 3 ***constrLiaison*** permet de créer la liaison entre le nœud du scaffold et le nœud de la référence correspondant qui viennent d'être créés mais ne renvoie rien. Enfin l'algorithme 1 ***Principal*** permet d'écrire le fichier DOT qui permettra d'encoder le graphe en prenant le fichier texte *résultat* en entrée. Cet algorithme 1 ***Principal*** utilise notamment les deux autres algorithmes 2 (***constrNoeud***) et 3 (***constrLiaison***).

Algorithme 1 : Principal

Résultat : Fichier DOT permettant d'encoder le graphe correspondant au fichier texte donné en entrée

Initialisation des variables globales statiques :

Axe_D ← **16**, **Axe_G** ← **0**, **Axe_M** ← **8**, *entiers* indiquant l'abscisse du milieu des scaffolds placés à droite et des scaffolds placés à gauche de la référence, ainsi que l'abscisse du milieu de la référence;

nb_color ← **3**, un *entier* indiquant le nombre de couleurs dans le tableau T;

T, un *tableau de chaînes de caractères* contenant les noms des couleurs de nœuds tel que **T=[color1,color2,color3]**;

couleur_arr ← **"gray55"**, *chaîne de caractères* indiquant la couleur du contour des nœuds arrières;

dec ← **0.1**, un *flottant* représentant le décalage entre deux nœuds successifs au sein d'un scaffold et de la référence;

Initialisation des variables locales : **L**, un *tableau d'entiers* qui contient les 5 éléments d'une ligne de texte;

Gauche et **Décal**, *deux booléens*, si Gauche=1 alors le scaffold sera placé à gauche de la référence, sinon à droite. Décal permet le petit décalage entre chaque bloc au sein d'un scaffold et de la référence, il est lié à la variable **dec**;

Deb_Sca, **Fin_Sca**, **Deb_Ref**, **Fin_Ref**, **Num_Sca**, **Num_Sca_Courant**, **Deb_Bloc_Sca**, **Fin_Bloc_Sca**, **Deb_Bloc_Ref**, **Fin_Bloc_Ref**, des *entiers*;

i, un *entier* permettant de gérer la couleur;

descr1 et **descr2**, des *entiers*, descr1 est le descripteur du fichier *résultat*, descr2 est le descripteur du fichier DOT;

j_1 et **j_2**, deux entiers représentant les numéros de nœuds;

DÉBUT

Ouverture des deux fichiers. Un en lecture (fichier *résultat*) et l'autre en écriture (fichier DOT);

```
// Traiter la première ligne de texte :
```

L ← ***Lirefic()***;

Num_Sca_Courant ← **L[0]**;

Deb_Sca ← **L[1]** ; **Fin_Sca** ← **L[2]**;

Deb_Ref ← **L[3]** ; **Fin_Ref** ← **L[4]**;

Gauche ← **Vrai** et **Décal** ← **Faux**;

i ← **0**;

while ***L est non vide*** **do**

  Num_Sca ← **L[0]**;

  Deb_Bloc_Sca ← **L[1]** ; **Fin_Bloc_Sca** ← **L[2]**;

  Deb_Bloc_Ref ← **L[3]** ; **Fin_Bloc_Ref** ← **L[4]**;

  if ***(Num_Sca*** **!=** ***Num_Sca_Courant)*** **then**

```
    // on entre dans la boucle lorsqu'on est à la fin du scaffold
```

    Num_Sca_Courant ← **Num_Sca**;

```
    // Construire le noeud arrière :
```

    constrNoeud(1,Gauche,Décal,Deb_Sca,Fin_Sca,couleur_arr,descr2);

    Deb_Sca ← **L[1]**;

    Gauche ←**!Gauche**;

  end

  Fin_Sca ← **L[2]** ; **Fin_Ref** ← **L[4]**;

```
  // Construire nœud scaffold :
```

  j_1 = constrNoeud(1,Gauche,Décal,Deb_Bloc_Sca,Fin_Bloc_Sca,T[i],descr2);

```
  // Construire nœud référence :
```

  j_2 = constrNoeud(0,Gauche,Décal,Deb_Bloc_Ref,Fin_Bloc_Ref,T[i],descr2);

```
  // Créer liaison entre les deux nœuds :
```

  constrLiaison(j_1,j_2,descr2);

  i ← **min(i+1;(i+1)mod nb_color)**;

  Décal ←**!Décal**;

  L ← ***LireFic(descr1)***;

end

```
// Construire nœud arrière du dernier scaffold :
```

constrNoeud(1,Gauche,Décal,Deb_Sca,Fin_Sca,couleur_arr,descr2);

```
// Construire nœud référence arrière :
```

constrNoeud(0,Gauche,Décal,Deb_Ref,Fin_Ref,couleur_arr,descr2);

Fermeture des deux fichiers;

FIN

Commentaires de l'Algorithme 1 (Principal) :

Ligne 4 : **T** ne varie jamais et on le parcourt en boucle pour attribuer aux nœuds chaque couleur les unes après les autres.

Ligne 9 :

- **Deb_Sca** et **Fin_Sca** représentent respectivement le début et la fin d'un scaffold
- **Deb_Ref** et **Fin_Ref** représentent respectivement le début et la fin de la référence
- **Num_Sca**
- **Num_Sca_Courant** représente le numéro du scaffold courant, correspondant aux chiffres de la première colonne du texte
- **Deb_Bloc_Sca** et **Fin_Bloc_Sca** représentent respectivement le début et la fin d'un nœud d'un scaffold
- **Deb_Bloc_Ref** et **Fin_Bloc_Ref** représentent respectivement le début et la fin d'un nœud de la référence

(voir la figure 21 ci-dessous)

Ligne 15 : *Lirefic()* est une fonction permettant de lire le fichier ligne par ligne.

Ligne 23 : La boucle **Tant que** est une boucle de lecture permettant de lire le texte ligne après ligne. On commence toujours cette boucle sur la ligne 1.

Ligne 27 : On entre dans la boucle **Si** si on est passé au scaffold suivant.

Ligne 37 : On passe à la ligne suivante dans le fichier texte.

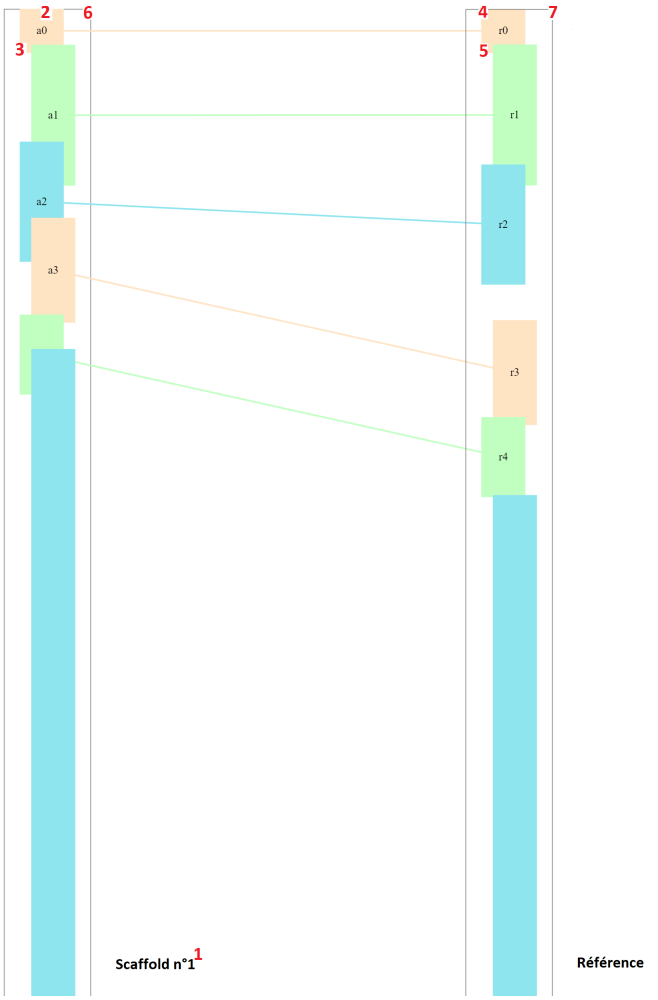

FIGURE 21 – Représentation des variables locales correspondant aux différents nœuds sur un exemple de graphe : le chiffre **1** correspond à **Num_Sca_Courant**, le **2** à **Deb_Bloc_Sca**, le **3** à **Fin_Bloc_Sca**, le **4** à **Deb_Bloc_Ref**, le **5** à **Fin_Bloc_Ref**, le **6** à **Deb_Sca** et le **7** à **Deb_Ref**

Algorithme 2 : constrNoeud

Données : **typeNoeud**, un *booléen*, si typeNoeud=1 c'est un nœud scaffold, un nœud référence sinon;
Gauche et **Décal** des *booléens*;
deb et **fin** des *entiers* représentant respectivement le début et la fin du nœud;
couleur, une *chaîne de caractères*, couleur qui permet de savoir de quel type de nœud il s'agit. Si couleur=couleur_arr, il s'agit d'un noeud arrière;
descr, un *entier*, descripteur de fichier;
Résultat : Renvoie le numéro du nœud construit et écrit dans le fichier DOT le code permettant de le créer

```
1  Initialisation des variables locales :
2  hauteur un entier ; milieu un flottant;
3  num un entier statique, numéro faisant office de nom de nœud;
4  DÉBUT
5  hauteur ← fin-deb ; milieu ← -((fin+deb)/2.);
   // pour la variable milieu, on choisi une valeur négative afin d'afficher les nœuds de
      haut en bas
6  num ← 0 ; num++;
7  if (couleur == couleur_arr) then
       // on crée un nœud arrière
8      if (!typeNoeud) then
           // sur la référence
9          écrire(descr,num+" [height="+hauteur+", width=1.5, pos=\""+Axe_M+","+milieu+"!\", style=empty,
             color="+couleur_arr+"];");
10     end
11     else
12         if Gauche then
               // sur un scaffold à gauche
13             écrire(descr,num+" [height="+hauteur+", width=1.5, pos=\""+Axe_G+","+milieu+"!\", style=empty,
                 color="+couleur_arr+"];");
14         end
15         else
               // sur un scaffold à droite
16             écrire(descr,num+" [height="+hauteur+", width=1.5, pos=\""+Axe_D+","+milieu+"!\", style=empty,
                 color="+couleur_arr+"];");
17         end
18     end
19 end
20 else
       // on crée un nœud bloc
21     if (!typeNoeud) then
           // sur la référence
22         if Décal then
23             écrire(descr,num+" [height="+hauteur+", pos=\""+Axe_M-dec+","+milieu+"!\", color="+couleur+"];");
24         end
25         else
26             écrire(descr,num+" [height="+hauteur+", pos=\""+Axe_M+dec+","+milieu+"!\", color="+couleur+"];");
27         end
28     end
29     else
30         if Gauche then
               // sur un scaffold à gauche
31             if Décal then
32                 écrire(descr,num+" [height="+hauteur+", pos=\""+Axe_D-dec+","+milieu+"!\", color="+couleur+"];");
33             end
34             else
35                 écrire(descr,num+" [height="+hauteur+", pos=\""+Axe_D+dec+","+milieu+"!\", color="+couleur+"];");
36             end
37         end
38         else
               // sur un scaffold à droite
39             if Décal then
40                 écrire(descr,num+" [height="+hauteur+", pos=\""+Axe_G-dec+","+milieu+"!\", color="+couleur+"];");
41             end
42             else
43                 écrire(descr,num+" [height="+hauteur+", pos=\""+Axe_G+dec+","+milieu+"!\", color="+couleur+"];");
44             end
45         end
46     end
47 end
48 retourner num;
49 FIN
```

Algorithme 3 : constrLiaison

Données : **num1**, **num2** deux *entiers*, les numéros des deux nœuds correspondants à lier;
descr, un *entier*, descripteur de fichier;
Résultat : Ne renvoie rien, écrit dans le fichier DOT le code permettant de créer la liaison entre un nœud scaffold et un nœud référence correspondant

```
1 DÉBUT
2 écrire(descr,"num1 - - num2;");
3 FIN
```

La doctorante Elisa Henrion-Gueneau a pu implanté les algorithmes afin de les utiliser dans le cadre de ses recherches. Le résultat est bien celui attendu (voir la figure 22 ci-dessous).

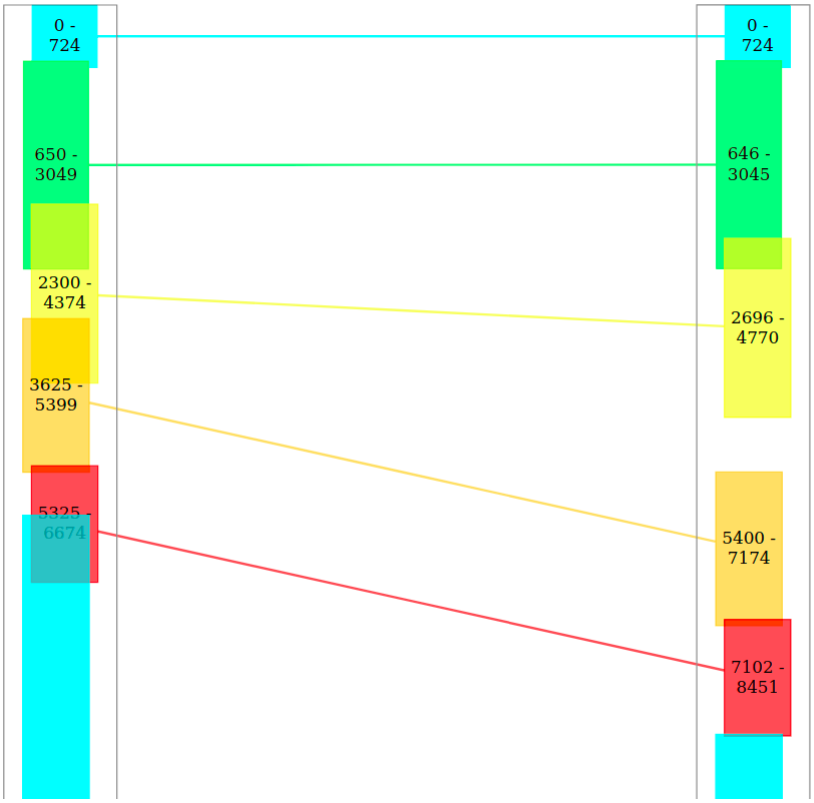

FIGURE 22 – Exemple d'application des algorithmes avec 5 couleurs

5 Conclusion

Ce stage m'a été très enrichissant car il m'a permis de faire mes premiers pas dans le domaine de la bioinformatique. Cela me conforte dans mes choix pour la suite de mes études, notamment pour le choix de master. De plus, étant mon premier stage de recherche en tant qu'étudiante à l'université, cela m'a permis de découvrir toutes les démarches nécessaires aussi bien sur le plan administratif que sur le plan pédagogique, avec le choix du sujet, les objectifs fixés, etc. En outre, j'ai pu mettre en application des connaissances que j'ai acquises durant mes deux années de licence, me permettant ainsi d'avoir une vision plus concrète des concepts vus en cours et de consolider mes acquis. J'ai aussi pu apprendre à rédiger correctement un rapport et à utiliser le langage ***LaTex***. De plus, j'ai appris à gérer une bibliographie avec ***BibTex*** et à écrire des algorithmes avec ***algorithm2e***, des outils qui me serviront énormément à l'avenir.

Références

[1] https://excalidraw.com/.

[2] https://dreampuf.github.io/GraphvizOnline.

[3] P. Alain Bernot. L'analyse des génomes - cartographie, séquençage, identification des gènes, 1996.

[4] S. W. Alex Cabral. The computer science behind dna sequencing. https://sitn.hms.harvard.edu/flash/2019/the-computer-science-behind-dna-sequencing/, 2019.

[5] CNRS. Le génome humain : de qui, pour qui, pourquoi? http://www2.cnrs.fr/sites/communique/fichier/8_genome_humain.pdf.

[6] M. Csurös. Assemblage *de novo*. http://www.iro.umontreal.ca/~csuros/IFT6299/H2014/content/prez13-assembly.pdf, 2011.

[7] C. Deluzarche. Pcr. https://www.futura-sciences.com/sante/definitions/genetique-pcr-91/.

[8] G. documentation. Color names. https ://graphviz.org/doc/info/colors.html.

[9] G. documentation. The dot language. https://graphviz.org/doc/info/lang.html.

[10] C. Fiorio. http://tug.ctan.org/macros/latex/contrib/algorithm2e/doc/algorithm2e.pdf, 1995-2017.

[11] GenoScreen. Assemblage - reconstituer fidèlement les données génomiques. https://www.genoscreen.fr/fr/genoscreen-services/bioinformatique/assemblage.

[12] J.-C. D. P. B. e. M. B. J. Lamoril, N. Ameziane. Les techniques de séquençage de l'adn : une révolution en marche. première partie. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7147846/, 2008.

[13] A. Louis. *Outils d'assemblage de génomes*. 2019.

[14] R. Paris. Une des plus grandes révolutions du vivant : l'émergence de la cellule eucaryote. http://www.matierevolution.fr/spip.php?article5135, 2019.

[15] F. Santé. Eucaryote. https://www.futura-sciences.com/sante/definitions/genetique-eucaryote-144/.

[16] F. Santé. Génome. https://www.futura-sciences.com/sante/definitions/genetique-genome-154/.

[17] SlideToDoc. Dna sequencing project dna sequencing. https://slidetodoc.com/dna-sequencing-project-dna-sequencing-how-we-obtain/.

[18] Wikipedia. Genbank. https://fr.wikipedia.org/wiki/GenBank.

[19] Wikipédia. Acide désoxyribonucléique. https://fr.wikipedia.org/wiki/Acide_désoxyribonucléique.

[20] Wikipédia. Clivage (chimie). https://fr.wikipedia.org/wiki/Clivage_(chimie).

[21] Wikipédia. Contig. https://fr.wikipedia.org/wiki/Contig.

[22] Wikipédia. Fastq. https://fr.wikipedia.org/wiki/FASTQ.

[23] Wikipédia. Graphviz. https://fr.wikipedia.org/wiki/Graphviz.

[24] Wikipédia. Maxam–gilbert sequencing. https://en.wikipedia.org/wiki/Maxam\T1\textendashGilbert_sequencing.

[25] Wikipédia. Pipéridine. https://fr.wikipedia.org/wiki/Pipéridine.

[26] Wikipédia. Scaffolding (bioinformatics). https://en.wikipedia.org/wiki/Scaffolding_(bioinformatics).