UMONS
Université de Mons

Faculté des Sciences

Assemblage de fragments d'ADN
Rapport de projet

Projet réalisé dans le cadre
de la 1er Master en Sciences informatique
pour le cours d'« Algorithmique et bioinformatique »

réalisé par

WATILLON	Thibaut	THIBAUT.WATILLON@student.umons.ac.be
OPSOMMER	Sophie	SOPHIE.OPSOMMER@student.umons.ac.be

Faculté
des Sciences

Sous la direction de: *Titulaire* : O. DELGRANGE
Assistant : D. MASLOWSKI

Année Académique 2014-2015

ACADÉMIE
UNIVERSITAIRE
WALLONIE-
BRUXELLES

Résumé

Ce *rapport* est rendu dans le cadre du cursus de première année de « Master en Sciences Informatiques »pour le cours de *Algorithmique et bioinformatique* (dont le titulaire est Mr. *O. Delgrange* et l'assistant est Mr. *D. Maslowski* en année académique 2014-2015) . Le but de ce rapport est de présenter les résultats de l'implémentation du projet.

Table des matières

1 Introduction

1.1 Problème posé

Ce projet consiste à développer un programme informatique qui a la particularité de pouvoir, à partir d'une collection de fragments (de longueur variable), créer un contig le plus petit possible avec tous les fragments (ou leur complémentaire inversé).

1.2 Les données

1.2.1 En entrée

Les fragments sont fournis par le biais d'un fichier au format « .FASTA ». La structure d'un tel format est tel que : un fichier contient au moins 1 fragment. Chaque fragment est de la forme :

```
> identifiant-de-la-séquence [commentaire]
suite-de-caractères
[suite-de-caractères]
[suite-de-caractères]
...
```

Notons que il doit-être possible d'introduire des fichiers simplifiés, c'est-à-dire des fichiers où le sens de lecture des fragments est connu. Pour ceux-ci, il ne faut donc par calculer les fragments complémentaires inversés.

1.2.2 En sortie

Les fichiers générés par le programme se trouvent à la racine du programme. Ils sont au format :

```
> Nom [noms-etudiants] Collection [num-collection] Longueur [longueur-sequence-cible]
suite-de-caractères
```

1.3 Etapes clés

Les étapes clés du projet sont les suivantes : (chronologiquement)

1. Compréhension du projet à réaliser
2. Lecture du fichier de fragments
3. Construction du graphe
4. Implémentation de l'algorithme Greedy
5. Construction de la super chaine
6. Ecriture du résultat dans un fichier

2 Implémentation

2.1 Structures de données

Lors de la lecture des fichiers, les fragments sont insérés dans des objets *Fragment*.

Ces objets *Fragment* sont ensuite insérés dans des objets *Node* correspondant aux noeuds du graph de chevauchement. Ces objets en plus de contenir un fragment contiennent aussi un booléen *in* et un booléen *out* nécessaire pour le *Greedy*, une référence vers le noeud de son complémentaire inversé et une liste des noeuds dont les fragments sont inclus au fragment de ce noeud.

Pour appliquer l'algorithme *Greedy* sur le graph, il faut tout d'abord que ce graph possède des arcs. Pour ce faire, un objet *Edge* a été créé et contient le noeud de début et le noeud de fin de cet arc ainsi qu'un objet *Alignment* contenant les informations de l'alignement entre le fragment du noeud source et le fragment du noeud destination.

L'objet *Alignment* comprend donc : Le type d'alignement (préfixe, suffixe, inclu), le coût de l'alignement, l'indice du début de l'alignement, l'alignement du fragment du noeud source et l'alignement du fragment du noeud destination.

La dernière structure de données concerne le graph de chevauchement. Il contient une liste des noeuds ainsi qu'une liste des arcs.

Les objects *Fragment*, *Node*, *Edge*, *Alignement*, *AlignementType* et *Graph* sont représenté à la figure 1.

2.2 Construction du graph de chevauchement

Cette partie de l'application est la plus gourmande en terme de ressources et consiste à créer les arcs du graph de chevauchement.

En effet, lorsque nous avons n fragments et que nous devons calculer les complémentaires inversés, $n * (n - 2)$ arcs sont à créer. Si les fragments complémentaires inversés ne sont pas à calculer, $n * (n - 1)$ arcs sont à créer.

Pour réduire le temps de calcul, nous appliquons donc le multithreading.

Deux contraites sont à respecter lors de la construction des arcs. La première est qu'il ne faut pas créer d'arc entre un noeud et son complémentaire inversé. La seconde étant qu'il ne faut pas créer un arc ayant comme source et destination le même noeud. D'où le $n * (n - 2)$.

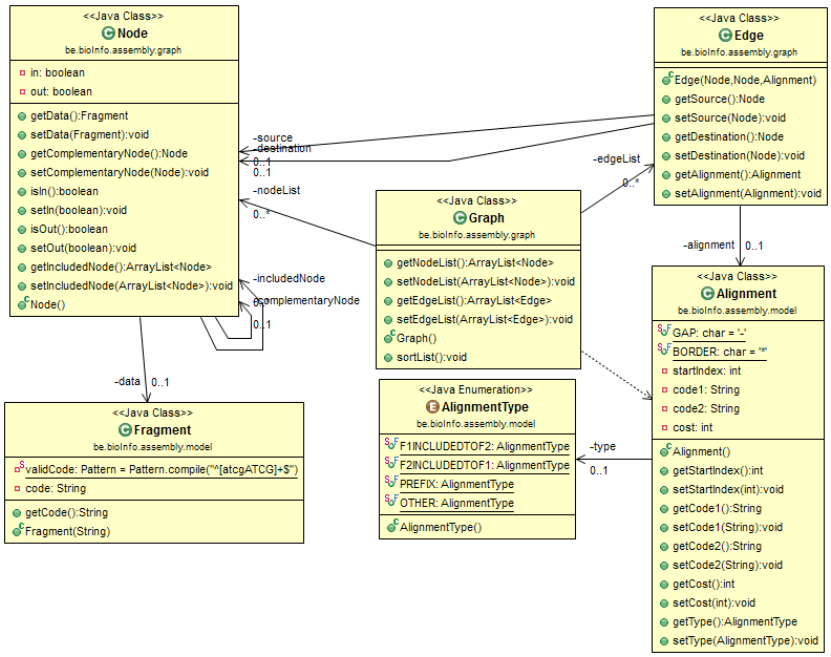

FIGURE 1 – Visualisation des objects principaux

La création des arcs introduit aussi l'algorithme de l'alignement semi-global.

2.3 Algorithme de l'alignement semi-global

Lors de la création des arcs, l'alignement entre les deux fragments concernés doit être calculé pour attribuer un coût à l'arc et pour déterminer le type d'alignement entre ces deux fragments.

Lors de la construction d'un arc de n_1 à n_2, la matrice calculée et utilisée pour déterminer l'alignement entre les deux fragments sera également utilisée pour déterminer l'alignement de l'arc allant de n_2 à n_1.

Pour construire l'alignement de n_1 à n_2, on se positionne sur la valeur maximale de la dernière ligne de la matrice tandis que pour construire l'alignement de n_2 à n_1, on se positionne sur la valeur maximale de la dernière colonne.

Une fois l'algorithme exécuté, nous obtenons deux chaînes de caractères représentant l'alignement entre les fragments :

ACTGCGTA****** : Fragment du noeud source (f1)

****CGTAAAA*** : Fragment du noeud destination (f2)

Grâce à ces deux chaînes de caractères, nous pouvons déterminer le type d'alignement qui sera nécessaire lors de l'exécution du *Greedy*

Il est également possible de déterminer la position du début de l'alignement qui sera utile lors de la construction de la séquence.

Dans l'exemple, le début de l'alignement se trouve à la $4^{\text{ème}}$ position dans la seconde chaîne.

Une fois les arcs construits et les caractéristiques des alignements connues, l'algorithme *Greedy* peut être appliqué sur les arcs du graph.

2.4 Algorithme Greedy

Avant d'appliquer le *Greedy*, les arcs doivent tout d'abord être triés par ordre décroissant en fonction de leur coût.

Pour ce faire, la méthode *Collection.sort()* est utilisés sur la liste des arcs.

Le problème lors de ce tri est que les arcs ayant le même coût ont des positions à chaque fois différentes lors de l'exécution de*Collection.sort()* ce qui fait que les arcs sélectionnés par *Greedy* varient à chaque exécution de l'application.

L'algorithme a dû être modifié pour gérer les fragments inclus.

Pour ce faire, lors du traitement d'un arc $n_1 -> n_2$, si n_1 est inclu à n_2, le noeud n_1 est inséré dans la liste des fragments inclus à n_2. De plus, il faut empécher de prendre par la suite le noeud n_1 et son complémentaire inversé en mettant ses booléen *in* et *out* à $TRUE$ et en insérant n_1 et son complémentaire inversé dans l'ensemble de n_2. Cela est uniquement faisable si le noeud n_1 n'est pas déjà un noeud destination d'un arc choisit par *Greedy* aussi non il serait impossible de reformer la séquence.

Mainenant, si n_2 est inclu à n_1, le noeud n_2 est inséré dans la liste des fragments inclus à n_1. Il faut également empécher de prendre par la suite le noeud n_2 et son complémentaire inversé en mettant ses booléen *in* et *out* à $TRUE$ et en insérant n_2 et son complémentaire inversé dans l'ensemble de n_1. Ici, cela est uniquement faisable si le noeud n_2 n'est pas déjà un noeud source d'un arc choisit par *Greedy*.

Concernant les arcs dont les fragments ne sont pas inclus, il faut empêcher le noeud source d'être le noeud source d'un autre arc et d'empêcher le noeud destination d'être le noeud destination d'un autre arc. Il faut également empêcher de prendre des arcs ayant comme noeuds les complémentaires

inversés des noeuds déjà choisi par *Greedy*. Cela se fait via les booléens *in* et *out*. Il ne faut également pas oublier de mettre tous ces noeuds dans le même ensemble (complémentaires inversés compris).

Une fois l'algorithme du *Greedy* terminé, la liste des meilleurs arcs est disponible pour construire la séquence.

2.5 Construction de la séquence

Le but est, pour une séquence de taille N, de créer N tableaux où le tableau numéro i contient le nombre de fois que les nucléotïdes A, C, T, G et les gaps sont présents à la i^{eme} position dans la séquence. A noter que les tableaux contiendront une cellule permettant de savoir si à la position i, nous sommes en présence ou non d'un gap. Cela permettra de propager les gaps vers le bas. La figure 2 illustre cette structure de donnée.

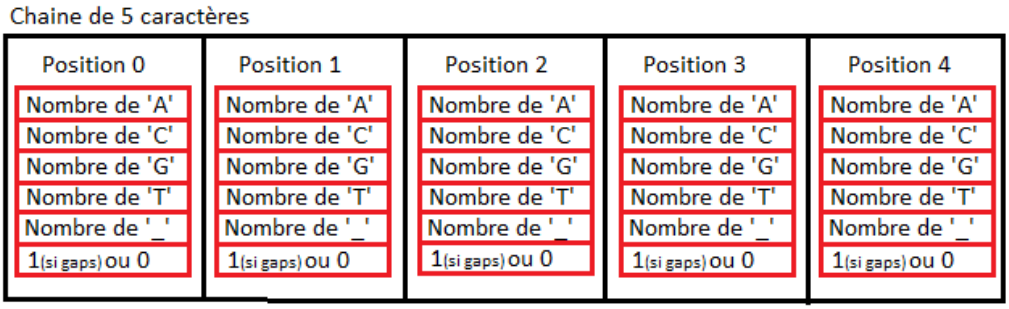

FIGURE 2 – Visualisation des tableaux

Une fois tous les arcs traités, pour reconstruire la séquence, il suffira de regarder dans chaque tableau quel est le symbole le plus représenté et ce symbole sera celui à insérer dans la séquence à la position i. A noter que pour empécher d'avoir le même nombre d'occurences de plusieurs symboles à la position i et ainsi effectuer un choix aléatoire, on utilise le score de l'alignement (coût de l'arc) comme valeur à insérer dans le tableau.

La première étape consiste à trouver l'arc débutant la séquence c'est-à-dire, l'arc dont le noeud source n'est la destination d'aucun arc et donc le noeud de destination est aussi la source d'un autre arc.Autrement dit, les booléen *in* et *out* du noeud source de cet arc doivent valoir respectivement $FALSE$ et $TRUE$ et ceux du noeud destination doivent valoir $TRUE$ et $TRUE$.

Une fois cet arc trouvé, on le traite et une fois cet arc traité, on cherche l'arc dont le noeud source équivaut au noeud destination de l'arc venant d'être traité pour ainsi former une chaîne.

Lors de la construction de la séquence, deux éléments importants doivent être pris en compte : la propagation des gaps vers les fragments précédents et la propagation des gaps vers les fragments futurs.

En effet, lors de la gestion d'un arc n_1->n_2, si la chaîne de caractères du fragment de n_1 possède un gap à la i^{eme} position cela signifie que le caractère $i+1$ de cette chaîne correspond au caractère à la position i dans la séquence. Il s'agit donc de décaler vers la droite le tableau correspondant au i^{eme} caractère de la séquence ainsi que tous les tableaux correspondant aux caractères se trouvant après le i^{eme} caractère et d'insérer un nouveau tableau à la position i. Un exemple de ce type de propagation de gaps est visible à la figure 3.

Un problème équivalent survient lorsque dans l'arc n_1->n_2, la chaîne de caractère du fragment de n_2 possède un gap à la i^{eme} position. Les futurs fragments à insérer dans la séquence devront faire correspondre leur i^{eme} caractère au $i+1^{eme}$ caractère de la séquence. Un exemple de ce type de propagation de gaps est visible à la figure 4.

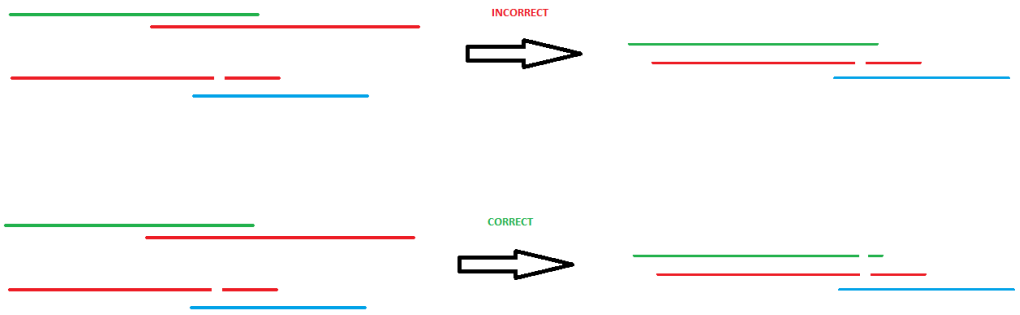

FIGURE 3 – Exemple de propagation d'un gap vers les fragments précédents

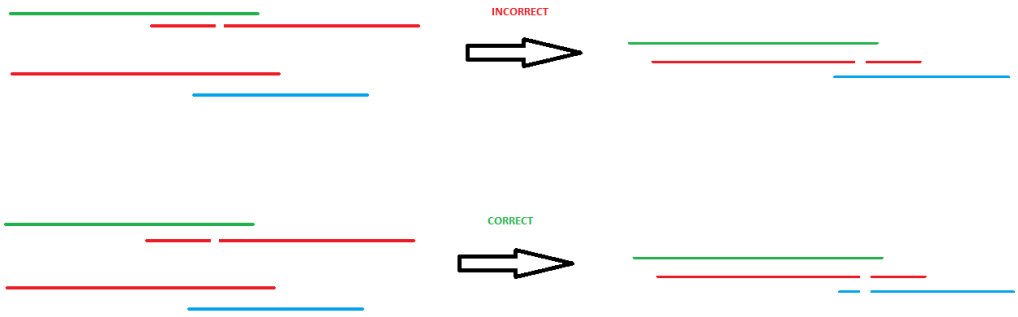

FIGURE 4 – Exemple de propagation d'un gap vers les fragments futurs

3 Résultats

Voici les résultats obtenus :

3.1 Collection 1

Les deux résultats présentés à la figure 5 sont les deux meilleurs résultats que nous ayons obtenu au cours de nos nombreux tests. On y voit que l'ensemble (ou presque) de la séquence cible a été retrouvée, cependant (tel que énoncé dans les points faibles), on observe aussi que de nombreux fragments n'ont pas été compris dans la partie 'cible' de la séquence (et ont été placé avant et après).

3.2 Collection 4

Les deux résultats présentés à la figure 6 sont les deux meilleurs résultats que nous ayons obtenu au cours de nos nombreux tests. On y observe que les résultats sont un peu moins bons que pour la collection 1. En effet ici, la partie cible est segmentée en plus de morceaux et ces morceaux sont plus éparpillés au sein du contig. Mais, notons tout de même que l'ensemble (ou presque) de la séquence cible a été retrouvée(bien que de nombreux fragments n'ont pas été compris dans la partie 'cible' de la séquence).

3.3 Collection 5

Les trois résultats présentés à la figure 7 sont nos meilleurs résultats pour cette collection. On y observe que les résultats sont encore un peu moins bons que pour la collection 4. En effet en plus d'une segmentation de la partie cible qui augmente, l'ensemble de la séquence cible n'est plus retrouvée.

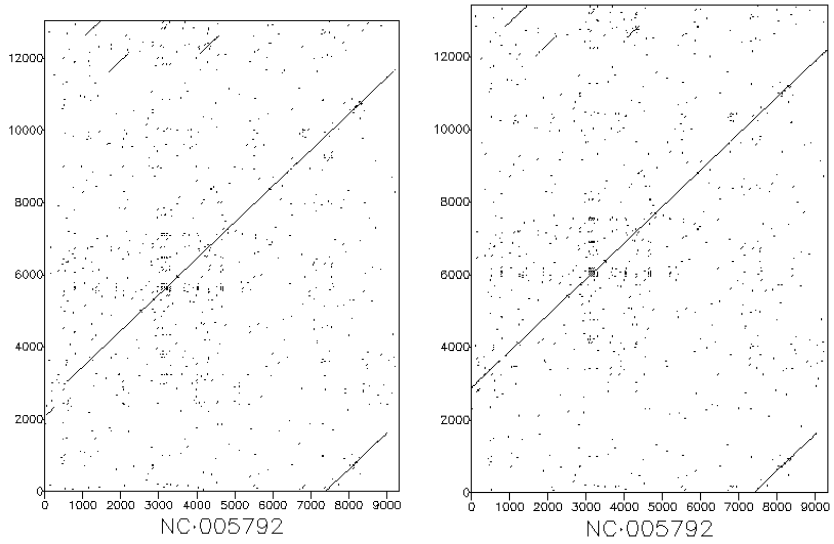

FIGURE 5 – Résultats de la comparaison entre les fichiers générés et le fichier cible de la collection 1

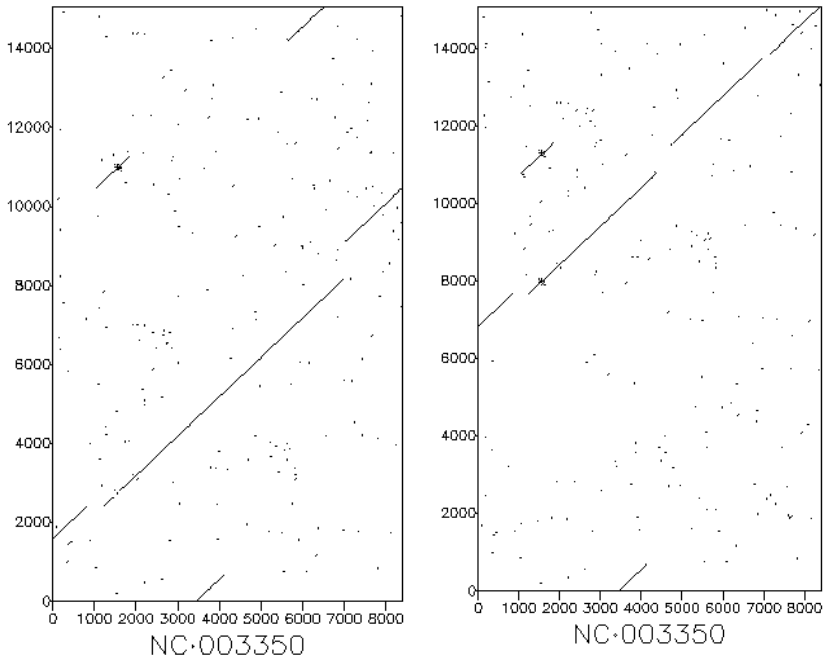

FIGURE 6 – Résultats de la comparaison entre les fichiers générés et le fichier cible de la collection 4

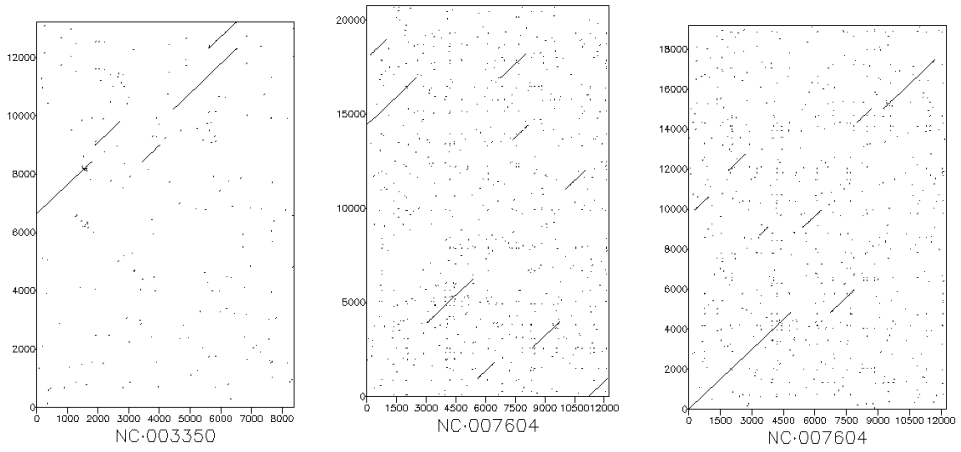

FIGURE 7 – Résultats de la comparaison entre les fichiers générés et le fichier cible de la collection 5

3.4 Collection 2

Les trois résultats présentés à la figure 8 sont nos meilleurs résultats pour cette collection. On y observe que les résultats sont mauvais. En effet aucune ligne ne se distingue, juste des points et des nuages un peu plus foncé par endroit.

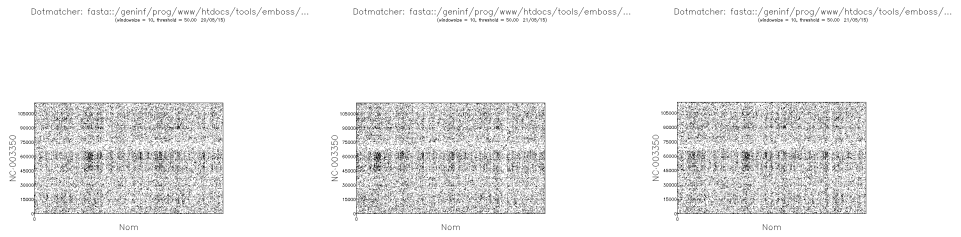

FIGURE 8 – Résultats de la comparaison entre les fichiers générés et le fichier cible de la collection 2

3.5 Collection S1

La collection S1 correspond aux mêmes fragments que la collection 1 seulement les fragments complémentaires et inversés ne doivent pas êtres calculés, ils sont fournis.

La figure 9 est notre meilleur résultat. Il est, de façon évidente, bien meilleur que toutes les autres comparaisons préalables. On y voit que l'en-

semble (ou presque) de la séquence cible a été retrouvée, et en plus, la longeur de notre résultat est comparable à la longueur de la séquence cible.

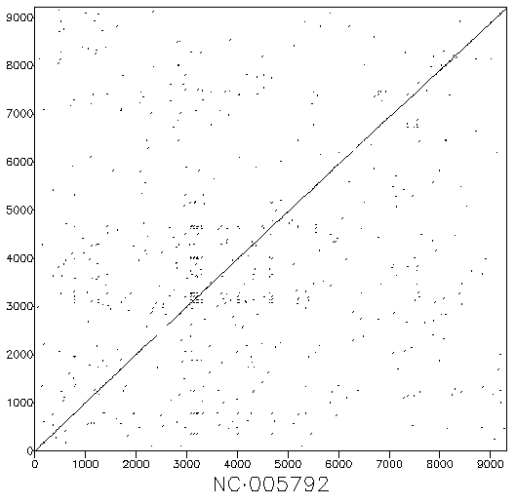

FIGURE 9 – Résultat de la comparaison entre le fichier généré et le fichier cible de la collection S1

3.6 Collection S2

La figure 10 est notre meilleur résultat. Il est, de façon évidente, bien meilleur que toutes les autres comparaisons préalables réalisée sur les fichiers non simplifié et comparable au résultat du fichier de la collection S1. On y voit que l'ensemble (ou presque) de la séquence cible a été retrouvée, et en plus, la longeur de notre résultat est comparable à la longueur de la séquence cible.

3.7 Collection S3

Pour la collection 3 simplifiées, 2 fichiers cibles ont été généré et sont disponible avec le programme seulement le site internet `http://emboss.bioinformatics.nl/cgi-bin/emboss/dotmatcher/` ne nous a pas permis d'obtenir une visualisation de la qualité du résultat.

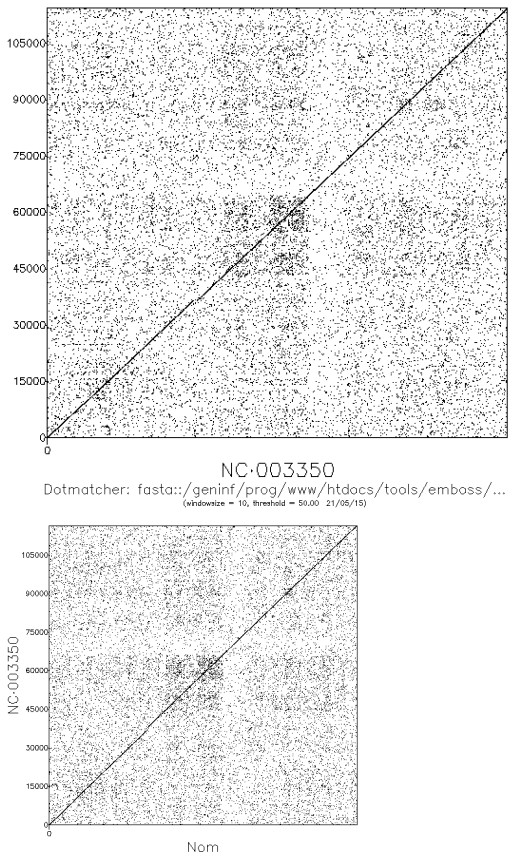

FIGURE 10 – Résultat de la comparaison entre le fichier généré et le fichier cible de la collection S2

4 Problèmes rencontrés

Nous avons rencontrés de nombreux problèmes et ce à chaque étape de la réalisation de l'application.

La première étant lors de l'implémentation de l'algorithme de l'alignement semi-global et la détermination du type de l'alignement. Nous avons longuement essayé de déterminer le type de l'alignement (préfixe, suffixe, inclusion) via la matrice de l'alignement mais cette méthode donnait des mauvais résultats lorsque l'alignement contenait des gaps. Nous nous sommes alors basé sur les chaînes de caractères représentant l'alignement et nous avons obtenus de meilleurs résultats.

La seconde difficulté était au niveau de l'algorithme *Greedy* et la gestion des fragments inclus. Il nous a fallu un certains temps avant d'arriver à les

gérer correctement.

La dernière difficulté concerne la construction de la séquence en elle même. Cette étape fut la plus dure car de nombreux éléments étaient à prendre en compte. Comment gérer les gaps, les fragments inclus, comment reconstituer une chaine via les arcs,...

5 Logiciel

5.1 Points forts

Un point fort du logiciel est qu'il est multithreadé au niveau de la construction des arcs du graph permettant ainsi de gagner un temps considérable.

Un seconde point fort est qu'il propose une interface graphique permettant d'aller directement sélectionner le fichier à traiter sans devoir insérer son chemin d'accès.

5.2 Points faibles

Un point faible du logiciel est la variabilité des résultats à chaque exécution de l'application. Cela vient du fait que lors du tri des arcs par ordre décroissant en fonction de leur coût, la méthode *Collection.sort()* trie différement les arcs ayant le même coût.

Un second point faible est que les fragments inclus n'ont pas été traités lors de la construction de la séquence par manque de temps bien qu'ils ont été détecté par *Greedy*.

5.3 Erreurs connues

Une erreur connue concerne la longueur de la séquence obtenue. Nous ne savons pas pourquoi la séquence contient beaucoup plus de caractère que la séquence cible.

Un second point à soulever concerne la barre de progression. En effet, ayant 4 étapes dans le processus de résolution, chaque étape terminée fait avancer la barre de 25%. Hors, pour une meilleur exactitude, elle devrait tenir compte du nombre d'arc et du fait que quand ce nombre augmente, le temps pour construire le graphe devient considérable et les 3 autres étapes deviennent négligeable en temps.

Finalement, l'analyse par le site `http://emboss.bioinformatics.nl/cgi-bin/emboss/dotmatcher/` ne nous permet pas de visualiser la qualité du fichier cible. Nous déclarons ne pas en connaître la raison.

6 Répartition des tâches

Tâche	Réalisé par Thibaut	Réalisé par sophie
Compréhension du projet à réaliser	50%	50%
Lecture du fichier de fragments	80%	20%
Construction du graphe	60%	40%
Implémentation de l'algorithme Greedy	55%	45%
Construction de la super chaine	80%	20%
Ecriture du résultat dans un fichier	60%	40%
Interface	10%	90%
Rédaction du rapport	50%	50%

7 Conclusion

Au travers de ce travail, nous avons appris plusieurs choses. Ce fut d'abord un beau rappel de nos cours de biologie de secondaire. Ensuite, il nous a fait prendre conscience que les sciences font et feront encore partie de notre quotidient et que l'informatique à tout autant à apprendres aux autres sciences que ce que les autres sciences ont à apprendre à l'informatique.

A travers ce problème de séquensage d'ADN, nous avons eu l'occasion de mettre en pratique les concepts vu au cours mais aussi ceux vus tout au long de notre cursus tel que l'importance de la complexité des algorithmes, la programmation paralléle, la qualité du code,...

Ce projet fut une dur épreuve parce que, en plus de devoir gérer plusieurs projets en même temps avec des groupes différents (et donc des disponibilités différentes), il a fallu être très attentif à l'ensemble de la problématique à tout moment. Chaque décision avait un impact conséquent sur l'ensemble du programme et une décision prise trop vite pouvait rendre inutile de longue heures de travail.

Enfin, ce projet est différents des autres, parce que pour une fois, malgré que le projet soit fini, il n'est tout de même pas satisfaisant. En effet, certains résultats sont bons, mais d'autres restent mauvais sans en comprendre la raison (et donc pas de possibilité d'amélioration).

Références

[1] O. Delgrange & D. Maslowski, Algorithmique et bioinformatique, Enoncé du projet, Jan 2015, UMONS .

[2] O. Delgrange, Cours d'Algorithmique et bioinformatique, 2014 - 2015, UMONS .

[3] O. Delgrange, Algorithmique et bioinformatique, Assemblage de fragments d'ADN - Projet (slides de présentation du projet), 2014 - 2015, UMONS .