UMONS
Université de Mons

Faculté des Sciences

# Plus courts chemins dans un graphe pondéré

Projet réalisé dans le cadre
de la 1er Master en Sciences informatique

réalisé par
Simon Olbregts

Faculté
des Sciences

Sous la direction de:
Véronique Bruyère

novembre 2014

ACADÉMIE
UNIVERSITAIRE
WALLONIE-
BRUXELLES

# Table des matières

**1 Introduction** **2**

**2 Implémentation** **2**
2.1 Structures de données . . . . . 2
2.2 Construction du graph de chevauchement . . . . . 2
2.3 Algorithme de l'alignement semi-global . . . . . 3
2.4 Algorithme Greedy . . . . . 3
2.5 Construction de la séquence . . . . . 4

**3 Résultats** **5**

**4 Problèmes rencontrés** **5**

**5 Logiciel** **7**
5.1 Points forts . . . . . 7
5.2 Points faibles . . . . . 7
5.3 Erreurs connues . . . . . 7

**6 Répartition des tâches** **8**

**7 Conclusion** **8**

# 1 Introduction

# 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.

## 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 $n * (n - 2)$

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 utilisé 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 de créer un tableau contenant le nombre de fois que les nucléotïdes A, C, T, G et le nombre de gaps sont présents à la $i^{eme}$ position dans la séquence et ce pour chaque position. Si la séquence fait 1000 caractères, nous aurons donc 1000 tableaux. A noter que les tableaux contiendrons 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.

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 est lié qu'au noeud destination et dont le noeud destination n'est lié qu'au noeud source de cet arc et à un noeud 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 propager le gap dans les fragments précédemment traiter pour que le $i^{eme}$ caractère de la séquence corresponde au $i + 1^{eme}$ caractère de la chaîne. Un exemple de ce type de propagation de gaps est visible à la figure 1.

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 2.

# 3 Résultats

# 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 essayer de déterminer le type de l'alignement (préfixe, suffixe, inclusion)

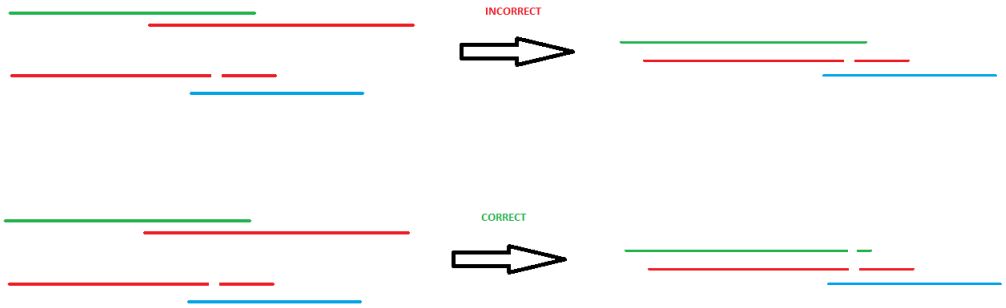


FIGURE 1 – Exemple de propagation d'un gap vers les fragments précédents

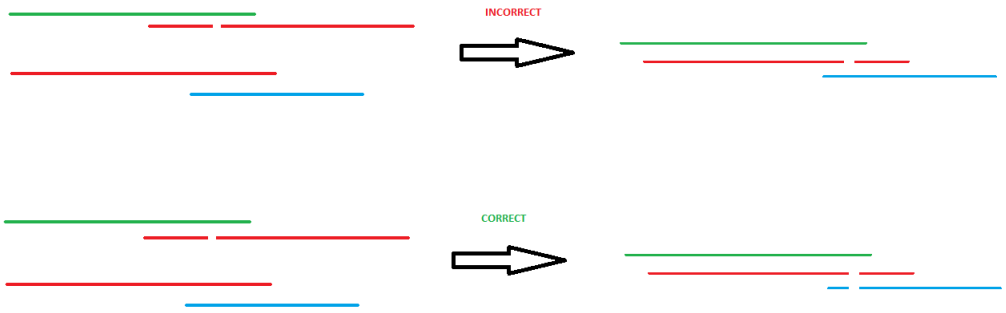


FIGURE 2 – Exemple de propagation d'un gap vers les fragments futurs

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.

# 6 Répartition des tâches

# 7 Conclusion