Projet d'IN301 : Comparaison de séquences d'ADN

Coline Gianfrotta et al.

October 24, 2019

L'ADN est le support de l'information génétique. Il est constitué de séquences de nucléotides représentés par les lettres A, C, G, T et est découpé en portions de séquences appelées gènes. La comparaison de ces gènes permet souvent de retrouver des liens de parenté entre les espèces possédant ces gènes.

Le but de ce projet sera alors de comparer des séquences d'ADN pour déterminer des familles de séquences, correspondant à des familles d'organismes.

Vous avez à votre disposition 20 fichiers contenant chacun une séquence d'ADN.
Votre travail va alors consister à :

- regrouper les séquences par famille en les comparant
- trouver la séquence "consensus" de chaque famille

1 Calculer la distance d'édition

Pour comparer deux séquences d'ADN, on va calculer la *distance d'édition* entre elles. Plus cette distance est grande, moins les séquences sont proches. Pour cela, on utilisera les valeurs de distance caractère par caractère, définies dans la Table 1.

d	A	C	G	T	$-$
A	0	2	1	2	1.5
C	2	0	2	1	1.5
G	1	2	0	2	1.5
T	2	1	2	0	1.5
$-$	1.5	1.5	1.5	1.5	

Table 1: Distance entre caractères, $-$ correspond à un blanc

On peut définir cette distance d'édition de deux manières, que vous devrez toutes deux implémenter dans votre programme. Elles sont présentées dans la suite par ordre de difficulté croissante.

On considère deux séquences $v = v_1.v_2....v_{lv}$ et $w = w_1.w_2...w_{lw}$. Chaque v_i $(0 \leq i \leq l_v)$ et w_i $(0 \leq i \leq l_w)$ correspondent à un caractère de la séquence et lv étant la longueur de la séquence v et lw celle de la séquence w.

D_1 : La façon la plus simple est de calculer la somme des distances caractère par caractère pour chaque position. On appelle cette distance D_1. Pour la séquence la plus courte, on ajoute des "blancs" en fin de chaîne pour pouvoir comparer avec les dernières positions de la séquence la plus longue. La distance associée à ces caractères spéciaux est définie en dernière ligne et dernière colonne de la Table 1 $(-)$. Formellement, on écrit donc :

$$D_1(v, w) = \sum_{1 \leq i < \max(lv, lw)} d(v_i, w_i)$$

Par exemple, la distance entre les séquences $ACGT$ et AGA vaudra
$D_1(ACGT, AGA) = d(A, A) + d(C, G) + d(G, A) + d(T, -) = 0 + 2 + 1 + 1.5 = 4.5$

D_2 : Mais il est en fait possible d'obtenir une valeur de distance plus faible en introduisant des "blancs" au sein des séquences (ce qu'on nommera insertion/délétion). On appellera cette deuxième distance D_2.

Par exemple, pour les séquences vues précédemment $ACGT$ et AGA, une meilleure valeur de distance peut être obtenue en faisant :
$D_2(ACGT, AGA) = d(A, A) + d(C, -) + d(G, G) + d(T, A) = 0 + 1.5 + 0 + 2 = 3.5$.
On introduit ici une insertion/délétion en 2ème position qui permet de diminuer la distance.
De façon plus visuelle, on peut représenter ces comparaisons entre séquences de la façon suivante :

$$\begin{array}{cc} AGA- & A-GA \\ | & | \ | \\ ACGT & ACGT \\ D_1 = 4.5 & D_2 = 3.5 \end{array}$$

Cette représentation est un *alignement* des deux séquences.

Ainsi, pour rechercher l'alignement donnant la distance d'édition la plus faible en autorisant des insertions/délétions, on pourrait tester toutes les insertions possibles pour toutes les positions, mais la complexité serait alors très grande. En revanche, si on raisonne de manière récursive, il existe une manière de découper le problème pour ne tester que certaines possibilités qui permettent d'obtenir la distance minimum entre deux séquences et le meilleur alignement. C'est une méthode de programmation dynamique.

Il faut considérer, pour cela, que pour chaque position, il a trois possibilités, et qu'on veut choisir celle qui augmente le moins la distance totale :

- soit on aligne les deux caractères
- soit on introduit un blanc dans la première séquence
- soit on introduit un blanc dans la deuxième séquence

Ces trois cas permettent d'écrire la formule générale suivante, pour calculer la distance d'édition entre les séquences v et w entre les positions 0 et i et 0 et j respectivement :

Soient les sous-chaînes :
$v_{[0,i]} = v_0.v_1....v_{i-1}.v_i$ avec $0 \leq i \leq lv$
$w_{[0,j]} = w_0.w_1...w_{j-1}.w_j$ avec $0 \leq j \leq lw$
alors la distance D_2 est définie récursivement par :

$$\begin{array}{rlcll} D_2(v_{[0,i]}, w_{[0,j]}) = \min(& D_2(v_{[0,i-1]} & , & w_{[0,j-1]}) & + \ d(v_i, w_j) \\ & D_2(v_{[0,i]} & , & w_{[0,j-1]}) & + \ d(v_i, -) \\ & D_2(v_{[0,i-1]} & , & w_{[0,j]}) & + \ d(-, w_j) \\ &) & & & \end{array}$$

Par exemple, si on reprend les deux séquences $ACGT$ et AGA :

$$\begin{array}{rl} D_2(ACGT, AGA) = & \min(D_2(AGC, AG) + d(T, A), \\ & \quad D_2(ACGT, AG) + d(T, -), \\ & \quad D_2(ACG, AGA) + d(-, A)) \end{array}$$

Calculer $D_2(v_{[0,lv]}, w_{[0,lw]})$ revient alors à calculer la distance d'édition totale entre v et w.

1.1 Travail à faire

1. Définir une structure de données permettant de stocker une séquence dans un type `SEQUENCE`.
2. Écrire les fonctions permettant de manipuler ces séquences (lire depuis un fichier, ...) et de faire du calcul dessus (calculer des distances, ...).

3. Fournir une structure de données `DISTANCE` pour stocker toutes les distances entre les paires de séquences contenues dans un répertoire.

   Ces types et fonctions devront être dans des fichiers `sequence.c`, `sequence.h`, `distance.c` et `distance.h`.

4. Écrire un fichier `main1.c` qui prend en argument un nom de répertoire et permettant de tester ce que vous avez programmé. Si vous avez besoin de fichiers supplémentaires n'hésitez pas.
5. Écrire un fichier `Makefile` et la commande `make partie1` devra compiler et exécuter.

2 Regrouper les séquences en familles

Dans la suite du projet (regrouper les séquences en famille et trouver la séquence consensus), vous utiliserez la distance d'édition la plus complète parmi celles que vous avez programmées (la deuxième si vous avez tout programmé, sinon la première).

Pour déterminer les familles de séquence, vous utiliserez l'algorithme suivant :

a. calculer les distances d'édition de toutes les séquences deux à deux et conserver les séquences associées à l'alignement dans de nouveaux fichiers (fait dans la partie 1)
b. rechercher la distance d'édition minimum D_{min} entre toutes les distances d'édition
c. rechercher la séquence s pour laquelle le nombre d'autres séquences ayant une distance d'édition de D_{min} avec elle est le plus grand
d. former la première famille avec cette séquence s et toutes les autres séquences qui ont une distance d'édition de D_{min} avec elle
e. recommencer à l'étape b. avec toutes les séquences qui n'ont pas encore été assignées à une famille

Une fois que vous avez obtenu les familles de séquence, vous devez ranger les fichiers correspondant aux séquences d'une même famille dans un même répertoire.

2.1 Travail à faire

6. Rajouter des fichiers `famille.c` et `famille.h` contenant les fonctions et les éventuelles structures de données nécessaires à la programmation de l'algorithme ci-dessus.
7. Programmer `main2.c` prenant en argument un nom de répertoire contenant les séquences ADN et affichant d'une manière compréhensible les familles.
8. Modifier le `Makefile` avec notamment une cible `partie2` permettant de compiler et d'exécuter cette partie

3 Trouver la séquence consensus de chaque famille

Dans une famille de séquences, on appelle *séquence consensus* une nouvelle séquence dans laquelle chaque caractère correspond à la lettre rencontrée le plus fréquemment à cette position dans la famille.

Vous devez trouver la séquence consensus associée à chacune des familles que vous avez déterminées. Pour cela, il faut :

a. aligner toutes les séquences d'une famille
b. puis, calculer la fréquence de chaque lettre en chaque position et en déduire ainsi la séquence consensus

Pour aligner toutes les séquences d'une famille, on commence par les deux séquences les plus proches, puis quand on a aligné n séquences $s_1, s_2, ..., s_n$ et que l'on veut rajouter la séquence s dans cet alignement, on cherche la séquence s_i la plus proche de s, puis on effectue l'alignement de s et s_i. Enfin on intègre s dans l'ensemble des alignements.

Exemple : On a l'alignement vu précédemment :

$$\begin{array}{c} A-GA \\ |\quad| \\ ACGT \\ D_2 = 3.5 \end{array}$$

On veut aligner une troisième séquence à ces deux séquences : $ACTG$. Cette dernière a une distance d'édition de 4 avec $ACGT$ et de 4.5 avec AGA. En effet :

$$\begin{array}{cc} ACTG & ACTG- \\ || & |\quad| \\ ACGT & A--GA \\ D_2 = 4 & D_2 = 4.5 \end{array}$$

On l'aligne donc avec ACGT:

$$\begin{array}{c} A-GA \\ |\quad| \\ ACGT \\ || \\ ACTG \end{array}$$

Pour la dernière position de cet alignement, chaque caractère est différent. Dans le cas où il n'y a pas de lettre qu'on trouve plus fréquemment que les autres à une position, on notera un '.' à cette position.
$ACG.$ est donc la séquence consensus de ces 3 séquences.

3.1 Travail à faire

9. Définir une structure de données `ALIGNEMENT` permettant de stocker l'alignement simultané de plusieurs séquences
10. Programmer les fonctions de manipulation de cette structure, en particulier l'initialisation avec deux séquences, l'ajout d'une séquence, le calcul de la séquence consensus.
11. Fournir les fichier `alignement.c` et `alignement.h` définissant les structures de données et les fonctions.
12. Programmer `main3.c` permettant de tester votre programme en prenant en argument un nom de dossier et affichant le plus proprement possible les familles et les séquences consensus.
13. Modifier le `Makefile` pour avoir une cible `make partie3`

4 Remarques finales

- Le projet est à faire en binôme Si vous faites le projet tout seul, aucun bonus ne sera accordé.
- Le projet sera à rendre sur moodle. le lien vous sera fourni ultérieurement. 3 points de moins pour le non respect de cette règle.
- Le projet est à rendre sous la forme d'une archive zip ou tar à l'exclusion de tout autre format. 3 points de moins pour le non respect de cette règle.
- L'archive devra s'appeler `NOM_NOM2.zip` ou `NOM1_NOM2.tar`, NOM1 et NOM2 étant les noms de famille des 2 membres du binôme. 3 points de moins pour le non respect de cette règle.
- L'archive devra contenir un dossier s'appelant `NOM1_NOM2`. 3 points de moins pour le non respect de cette règle.
- Le dossier devra contenir uniquement les fichiers sources et les séquences ADN et ne devra pas contenir les exécutables. 3 points de moins pour le non respect de cette règle.

- La date de remise sera après les vacances de Noël.

Quelques conseils :

1. Avoir une cible `clean` dans le `Makefile`
2. Avoir une cible `archive` qui crée le zip ou le tar au bon format.
3. Réfléchissez à la manière de stocker les familles dans un fichier ou plusieurs fichiers

Travail à faire éventuellement en plus :

- Écrire un programme `distance` qui à partir de deux fichiers de séquences donnés et du type de distance d'édition donnés en arguments, affiche l'alignement et la distance d'édition entre les deux séquences
- Écrire un programme qui sans arguments, liste les familles connues et, avec en argument une famille, affiche toutes les séquences qu'elle contient, ainsi que la séquence consensus associée