Algorithmique 3I003

Projet : Analyse de séquences en bioinformatique

Andréa KOSTAKIS

Licence Informatique L3
Univeristé Pierre et Marie Curie

Année universitaire 2016-2017

Première partie

Partie théorique

1 Alignement de coût minimal de séquences ADN : Conception et analyse des algorithmes

1.1 Algorithme naïf

Considérons un algorithme naïf qui consiste à énumérer tous les alignements possibles entre deux séquences X et Y de longueur d. Le principe de cette algorithme sera le calcul des alignements possibles pour toutes les sous-séquences de X et Y.

Algorithme 1 : EnumSeqNaïf

```
Données : X : séquence, Y : séquence, d : longueur
Résultat : Renvoie tous les alignements possibles
pour i de 1 à d faire
    pour toutes les sous-séquences de X faire
        pour toutes les sous-séquences de Y faire
            Aligner chaque n-ième symbole de la sous-séquence X avec
            ceux de Y.
            Enumérer les différents cas.
        fin
    fin
fin
retourner Enumeration
```

Complexité

Le nombre d'opérations T_N (T_N : étant la fonction complexité pire-cas pour l'algorithme) à effectuer par l'algorithme EnumSeqNaïf est :

$$T_N \geq 2^{2^d}, \text{ donc } T_N \in O(2^n)$$

Remarque

Le nombre N d'alignements possibles pour deux séquences de même longueur est donné par :

$$N = \sum_{k=0}^{n} C_{n+k}^{k} + k.C_{n}^{k}$$

1.2 Démonstration

On considère pour la suite de l'analyse deux séquences $X = (x_1, ..., x_m)$ et $Y = (y_1, ..., y_n)$.
Soit M un alignement de X et Y. Montrons que si $(x_m, y_n) \notin M$, alors x_m ou y_n n'apparaît pas dans M.
(Par l'absurde) Soit $(x_m, y_n) \notin M$, supposons que x_m et y_n apparaissent dans M.
x_m et y_n sont les derniers éléments des séquences X et Y respectivement et s'ils apparaissent tous les deux dans l'alignement M alors :

$$\exists (x_m, y_q), 1 \leq q < n \tag{1}$$

et

$$\exists (x_k, y_n), 1 \leq k < m \tag{2}$$

D'après la définition d'un alignement (pas de croisements) :
(1) , (2) => si $i < n$ alors $m < k$ (resp. $n<q$ si $k < m$). CONTRADICTION : car m et n sont les longueurs des deux séquences. Donc (1) ou (2), donc x_m ou y_n n'apparaît pas dans M.

1.3 Cas pour x_m et y_n

Supposons que l'on ne permet pas de correspondances $\begin{pmatrix} - \\ - \end{pmatrix}$ entre les séquences, car elles conduisent forcement à des solutions dominées. En se basant donc, sur la propriété de la question 1.2, on peut en déduire pour les derniers éléments des deux séquences X et Y, x_m et y_n respectivement :

a) $\begin{pmatrix} x_m \\ - \end{pmatrix}$: x_m peut-être présent que seul si $(x_m, y_n) \notin M$.

b) $\begin{pmatrix} x_m \\ y_n \end{pmatrix}$: si les deux éléments sont présents, ils sont forcement ensemble.
c) $\begin{pmatrix} - \\ y_n \end{pmatrix}$: y_n peut-être présent que seul si $(x_m, y_n) \notin M$.

1.4 Coût minimal

Soit $F(i,j)$ le coût minimal pour l'alignement des séquences $(x_1, ..., x_i)$ et $(y_1, ...y_j)$.
a) x_m est en correspondance avec un *gap* : $F(m,n) = F(m-1,n) + \delta_{gap}$
b) si $(x_m, y_n) \in M$: $F(m,n) = F(m-1,n-1) + \delta_{x_m y_n}$
c) y_m est en correspondance avec un *gap* : $F(m,n) = F(m,n-1) + \delta_{gap}$

1.5 Formule de récurrence

D'après les questions précédentes on peut déduire la formule de récurrence du coût minimal pour l'alignements de deux séquences, d'après les trois cas observés.
pour $i \geq 1, j \geq 1$:

$$F(i,j) = min \begin{cases} F(i-1,j) + \delta_{gap} \\ F(i-1,j-1) + \delta_{ij} \\ F(i,j-1) + \delta_{gap} \end{cases}$$

1.6 Cas de base

Montrons que $F(i,0) = i\delta_{gap}$ pour tout $i \in \{1, ..m\}$ et $F(0,j) = j\delta_{gap}$ pour tout $j \in \{1, ...n\}$.
$F(i,0)$ (respectivement $F(0,j)$) correspond au coût minimal de l'alignement de la séquence X (resp. Y) avec une séquence vide. Par définition, tous les éléments de la séquence X (resp. Y) seront alignées avec des *gaps* dans tous les cas, autant de *gaps* que la longueur respective des deux séquences. Cette alignement M_0 est unique, donc minimal pour tout $i \in \{1, ...m\}$ (resp. pour tout $j \in \{1, ...n\}$). Par définition du coût d'un alignement :

$$F(i,0) = f(M_0) = \sum_{x_i, y_i \in M_0} \delta_{x_i, y_j} + \sum_{x_i \notin M_0} \delta_{gap} + \sum_{y_j \notin M_0} \delta_{gap} = 0 + \sum_{x_i \notin M_0} \delta_{gap} + 0$$
$$= i\delta_{gap} \text{ pour tout } i \in \{1, ..m\}.$$

(recip. de la même manière $F(0,j) = j\delta_{gap}$ pour tout $j \in \{1, ...n\}$.)

1.7 Algorithme COUT1

Algorithme 2 : COUT1

Données : X : séquence, Y : séquence
Résultat : Renvoie la valeur d'un alignement de coût minimal pour les séquences X et Y.

$F(0,0) \leftarrow 0$
$m \leftarrow taille(X)$
$n \leftarrow taille(Y)$
pour i *de* 1 *à* m **faire**
  $F(i,0) \leftarrow i.\delta_{gap}$
fin
pour j *de* 1 *à* n **faire**
  $F(0,j) \leftarrow j.\delta_{gap}$
fin
pour i *de* 1 *à* m **faire**
  pour j *de* 1 *à* n **faire**
    $a \leftarrow F(i-1,j) + \delta_{gap}$
    $b \leftarrow F(i-j,j-1) + \delta_{ij}$
    $c \leftarrow F(i,j-1) + \delta_{gap}$
    $F(i,j) \leftarrow min(a,b,c)$
  fin
fin
retourner $F(m,n)$

Complexité en temps/espace

La complexité temporelle de COUT1 est en $O(n \times m)$, l'espace mémoire demandé pour l'exécution est un tableau de deux dimensions ($n \times m$).

1.8 Algorithme SOL1

Algorithme 3 : SOL1

Données : F : tableau à 2 dimensions des valeurs des couts $F(i,j)$ des séquences X et Y
Résultat : Renvoie l'alignement optimal des séquences X et Y

```
F(0,0) ← 0
m ← taille(X)
n ← taille(Y)
SOL ← ( )
pour i de 1 à m faire
    pour j de 1 à n faire
        a ← F(i−1, j) + δ_gap
        b ← F(i−j, j−1) + δ_ij
        c ← F(i, j−1) + δ_gap
        tmpMin ← min(a, b, c)
        si tmpMin == F(i−1, j) + δ_gap alors
            retourner SOL.ajouter({x_i, −})
        fin
        si tmpMin == F(i−1, j−1) + δ_{i,j} alors
            retourner SOL.ajouter({x_i, y_j})
        fin
        sinon
            SOL.ajouter({−, y_j})
        fin
    fin
fin
retourner SOL
```

Complexité et conclusion de la première approche

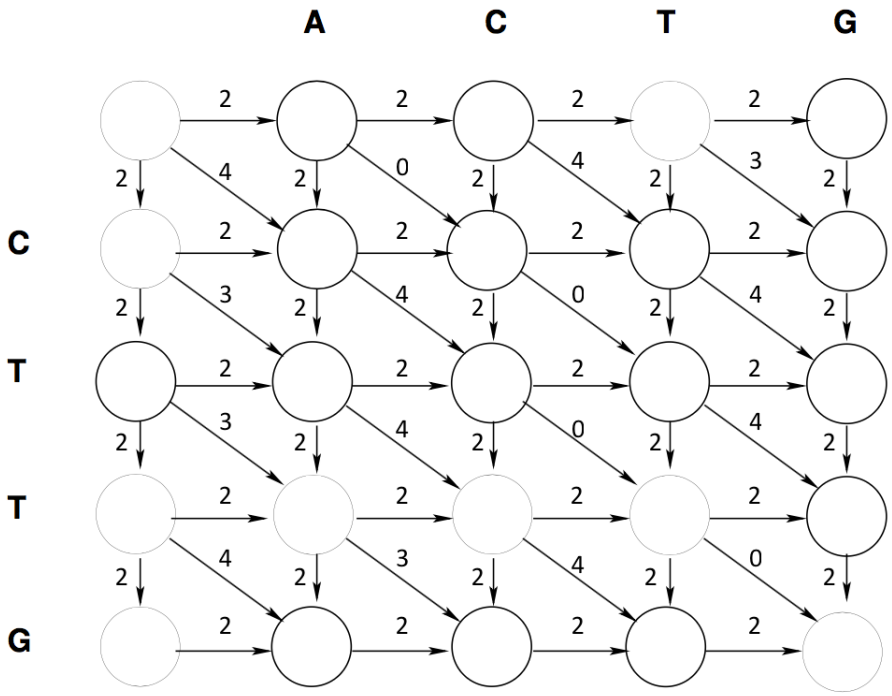
L'algorithme est en $O(m \times n)$, puisqu'il accède au éléments de F(i,j) un par un, puis effectue des opérations élémentaires. Pour conclure, cette approche s'effectue en deux temps ; elle nécessite la création d'une matrice de coût $F(i,j)$ en $O(m \times n)$, puis ensuite le traitement de cette dernière en $O(m+n)$ pour construire un alignement optimal des séquences données. Les séquences étant souvent très grande, la taille $(m * n)$ demandée par ces algorithme est assez imposante et les calculs effectuées peuvent être très couteux sur les deux temps d'exécution. Cette approche ne paraît pas donc la plus

optimale mais reste néanmoins une solution bien meilleure que l'approche naïve.

2 Représentation du problème d'alignement optimal sous forme de graphe

On s'intéresse dans cette partie à une représentation du problème sous forme de graphe.

2.1 Coût des arcs pour les séquences $X = (C, T, T, G)$ et $Y = (A, C, T, G)$



2.2 Longueur du plus court chemin et alignement optimal des séquences X et Y.

Montrons par récurrence forte sur $i + j$ que pour tout couple (i, j), $i \in \{0, ...\}$ et $j \in \{0, ...n\}$, on a $F(i, j) = g(i, j)$.

BASE : pour $i = 0, j = 0$ $(i + j = 0)$
On a bien :

$$F(0,0) = g(0,0) = 0$$

Par définition de $F(0,0)$ et du plus court chemin $g(0,0)$ (de $(0,0)$ à $(0,0)$).
INDUCTION :
Supposons que $F(k,q) = g(k,q)$ pour tous les k, $0 \leq k \leq i-1$ et pour tous les q, $0 \leq q \leq j-1$. Montrons que $F(i,j) = g(i,j)$

$$\text{par définition, } F(i,j) = min \begin{cases} F(i-1,j) + \delta_{gap} \\ F(i-1,j-1) + \delta_{i,j} \\ F(i,j-1) + \delta_{gap} \end{cases}$$

$$\text{par hypothèse de récurrence, } F(i,j) = g(i-1,j-1) + min(\delta_{gap}, \delta_{i,j})$$

Donc, par définition du plus court chemin, $g(i-1,j-1)+min(\delta_{gap}, \delta_{i,j}) = g(i,j)$

$$\text{donc, } F(i,j) = g(i,j)$$

Donc, pour tout couple $(i,j), i \in \{0,...\}$ et $j \in \{0,....n\}$, on a $F(i,j) = g(i,j)$. On peut en déduire que $F(m,n) = g(m,n)$ et donc que le coût d'un alignement optimal des séquences X et Y est la la longueur du plus d'un plus court chemin entre le sommet $(0,0)$ et (m,n) dans G_{XY}.

2.3 Algorithme de plus court chemin

On utilise ici l'algorithme de Dijktrsa pour déterminer le plus court chemin de $(0,0)$ à (m,n) dans G_{XY} . Ainsi, en l'implémentant avec une structure de tas, on obtient le plus court chemin avec une complexité en :

$$O((m \times n)\log(n))$$

Dans l'exemple 2.1, on obtient comme coût d'un l'alignement optimal $L = 4$ qui correspond à l'alignement $M = \{(-,A),(C,C),(T,T),(-,T),(G,G)\}$.
Remarque : ce n'est pas le seul alignement optimal.

2.4 Conclusion

En représentant le problème sous forme de graphe on obtient un alignement optimal de deux séquences X et Y avec une complexité $O((m \times n)log(n))$ en une exécution, bien meilleure que la solution de la partie précédente, qui pour le même résultat nécessite l'exécution de deux algorithme de complexité $O(m \times n)$ chacun.

3 Amélioration de la complexité spatiale des algorithmes de la partie 1

3.1 RAM - Longueur maximale

Dans les parties précédentes, la complexité spatiale des algorithmes étudiés est en $O(n*m)$, considérons ici que les gènes ont la même taille ($m = n$), qu'une mémoire vive d'ordinateur varie de 8 à 32GO et que le codage d'un caractère demande 1 octet. Soit D_{MAX} la longueur maximale (en nombre de nucléotides) que l'on peut traiter par les méthodes des parties précédentes.

$$8 \times (1024)^3 \leq m \times n \leq 32 \times (1024)^3$$
$$8 \times (1024)^3 \leq D_{MAX}^2 \leq 32 \times (1024)^3 \text{ , car } (m = n)$$

$$\text{Donc, } 92682 \leq D_{MAX} \leq 185363 \text{ (en nucléotides)}$$

3.2 Algorithme COUT2

Algorithme 4 : COUT2

Données : les séquences X et Y, $i \in \{1..m\}$ et $j \in \{1...n\}$
Résultat : Renvoie $F(i, j)$
pour l *de* 0 *à* j **faire**
  $ligneA[l] \leftarrow= l.\delta_{gap}$
fin
pour k *de* 1 *à* i **faire**
  $ligneB[0] \leftarrow k.\delta_{gap}$
  pour l *de* 1 *à* j **faire**
    $ligneB[l] \leftarrow$
    $min(ligneA[l-1] + \delta_{i,j}, ligneA[l] + \delta_{gap}, ligneB[k-1] + \delta_{gap})$
  fin
fin
retourner $ligneB[j]$

Complexité

Complexité temps : $O(i \times j)$, complexité espace : $O(i + j)$.

3.3 Algorithme COUT2BIS

On rappelle qu'on a montré (2.2) que $F(i,j) = g(i,j)$ qui est la longueur du plus court chemin du sommet $(0,0)$ au sommet (i,j) dans le graphe G_{XY}. On note à présent $h(i,j)$ la longueur du plus court chemin du sommet (i,j) au somment (m,n) dans le graphe G_{XY}.

Algorithme 5 : COUT2BIS

Données : les séquences X et Y, $i \in \{1..m\}$ et $j \in \{1...n\}$
Résultat : Renvoie $h(i,j)$: la valeur du plus court court chemin $(i,j) \rightarrow (m,n)$

$m \leftarrow X.longueur$
$n \leftarrow Y.longueur$
pour l *de* 0 *à* $n-j$ **faire**
  $ligneA[l] \leftarrow= l.\delta_{gap}$
fin
pour k *de* 1 *à* $(m-i)$ **faire**
  $ligneB[0] \leftarrow k.\delta_{gap}$
  pour l *de* 1 *à* $(n-j)$ **faire**
    $ligneB[l] \leftarrow$
    $min(ligneA[l-1]+\delta_{k+i,l+j}, ligneA[l]+\delta_{gap}, ligneB[l-1]+\delta_{gap})$
  fin
fin
retourner $ligneB[n-j]$

3.4 Plus court chemin passant par (i,j)

Dans le graphe G_{XY} :
la valeur du plus court chemin du sommet $(0,0)$ au sommet (i,j) est donnée par $g(i,j)$. La valeur du plus court chemin du sommet (i,j) au sommet (m,n) est donnée par $h(i,j)$.
On obtient donc la valeur du plus court chemin du sommet $(0,0)$ au sommet (m,n) en passant par (i,j) d'après : $g(i,j) + h(i,j)$. On peut en déduire la méthode pour calculer la valeur de ce plus court chemin en passant par (i,j) en utilisant les deux algorithmes précédents. Donc la valeur recherché est : COUT2(X,Y,i,j) + COUT2BIS(X,Y,i,j).

3.5 Algorithme SOL2

La complexité spatiale de l'algorithme SOL2$(0, 0, m, n)$ est en $O(n + m)$, pour le calcul de i^* l'algorithme fait l'appel de COUT2 et COUT2BIS qui sont eux en complexité spatiale de $O(n + m)$ au total pour leur exécution.