Travaux Pratiques : Gènes, génomes et alignement de séquences

1 Alignement de séquences (principes)

Au cours de l'évolution biologique, une séquence d'ADN change, le plus souvent en raison d'erreurs aléatoires dans la réplication ou la copie d'une séquence d'ADN. Ces erreurs de réplication sont appelées mutations. Les différents événements de mutation qui peuvent survenir sont:

- Des substitutions : où une base (ou un acide aminé, dans les séquences protéiques) est remplacée par une autre;
- Des insertions : lorsqu'une ou plusieurs bases contiguës sont insérées dans une séquence;
- Et des délétions : lorsqu'une ou plusieurs bases contiguës sont supprimées d'une séquence.

L'objectif de l'alignement de séquences est, à partir de deux séquences, de générer une hypothèse sur les positions de séquences dérivées d'une position de séquence ancestrale commune. En pratique, nous développons cette hypothèse en alignant les séquences les unes sur les autres en insérant des gaps, si nécessaire, d'une manière qui maximise leur similarité. C'est une approche de parcimonie maximale, où nous supposons que l'explication la plus simple (celle impliquant le moins d'événements de mutation extrême) est la plus probable.

L'alignement de séquences sert, entre autres, à identifier des fonctions de proteines. Le rôle des proteines étant essentiel, on part du principe que deux proteines de séquenses similaires auront des propriété similaires. On peut donc rapprocher la séquence de proteines inconnues à celles connues et en déduire leur fonction.

On représente un alignement de la manière suivante :

AGCTGCTATGATACCGACGAT
A - - T - C - AT -ATACCGACGAT

2 Approche naïve

Nous utiliserons une matrice d'identité, nous attribuons 1 point pour un match (même acide aminé) et 0 pour une substitution, une insertion/délétion.

Nous comparons deux séquences :
Seq1 : ACCGGTGGAACCGGTAACACCCAC
Seq2 : ACCGGTAACCGGTTAACACCCAC

La première étape est de créer une matrice de substitution, voir page 2. La seconde étape est de combler cette matrice en suivant les directives indiquées (match=1, autre = 0).

	A	C	C	G	G	T	G	G	A	A	C	C	G	G	T	T	A
A	1								1	1							1
C		2															
C																	
G																	
G																	
T																	
A																	
A																	
C																	
C																	
G																	
G																	
T																	
T																	
A																	

La matrice se remplit de la façon suivante :

- On part de la case en haut à gauche.
- Un déplacement en diagonal indique un match (+1)

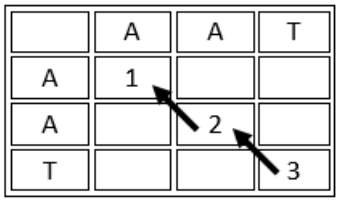
Une fois remplie, nous cherchons les plus longs chemins. C'est à dire que nous cherchons les endroits dans la matrice qui maximisent les déplacements en diagonale. Il faut alors repérer les nombres les plus grands dans la matrice.

Une fois identifiés, nous partons de la case la plus en bas à droite de la diagonale et nous la remontons en arrière pour transcrire l'alignement en écrivant les caractères de chacune des deux séquences à chaque ligne et colonne correspondant à la diagonale que vous suivez.

Lorsque nous rencontrons une coupure dans la diagonale, nous trouvons la diagonale suivante la plus longue qui commence dans une cellule qui est en haut et/ou à gauche de la cellule actuelle. Pour chaque déplacement vers le haut (non en diagonale), vous devez insérer un espace dans la séquence de l'axe horizontal de votre matrice. Pour chaque déplacement vers la gauche, insérez un espace dans la séquence de l'axe vertical de votre matrice.

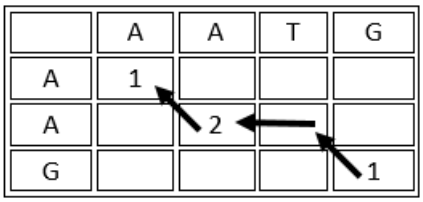
Exemple :

- Déplacement en diagonal = les lettres se font face (match ou mismatch)



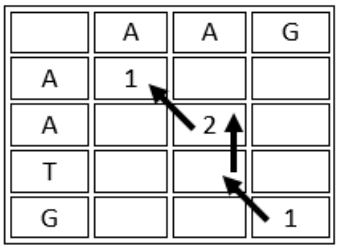
Alignement :
AAT
AAT

- Déplacement vers le haut = insertion d'un '-' dans la séquence horizontale.



Alignement :
AA-G
AATG

- Déplacement vers la droite = insertion d'un '-' dans la séquence verticale.



Alignement :
AATG
AA-G

Exercice 1 :

- Remplissez la matrice page 2, vous pouvez juste indiquer les matchs dans la matrice.
- Identifiez les plus longs chemins.
- Ecrire le meilleur alignement.
- La matrice utilisée est-elle la plus réaliste ? Justifiez.

3 Needleman-Wunch

L'alignement que nous venons de construire était un alignement global naïf, ce qui signifie que nous alignons les deux séquences de leur début à leur fin. Saul B. Needleman et Christian D. Wunsch en 1970 ont publié un algorithme d'alignement global, moins naïf, que nous allons programmer sur Java. Leur algorithme divise le problème global d'alignement en une série de problèmes de taille inférieure.

Il est aussi connu sur les noms : *optimal matching algorithm* et *global alignment technique*.

3.1 Principe

Nous comparerons deux séquences :
Seq1 = ATCGAACCTG
Seq2 = ATGAATG

a) Définition du coût des différents événements :

- Match = 3
- Mismatch = -1
- Indel (insertion ou delétion)= -2

Exercice 2 :

- Créez un package **align** dans Eclipse.
- Créez une nouvelle classe **alignement**.
- Importez le package **java.util.Arrays**.
- Créez trois variables static correspondant aux match, mismatch et indel.
- Attribuez-leur les valeurs correspondantes.
- Créer votre **main** de la façon suivante :

Java

```
1 public static void main(String[] args){
2       String S = "ATCGAACCTG";
3       String T = "ATGAATG";
4 }
```

b) Création de la matrice.

Contrairement à la matrice naïve les matrices de Needleman-Wunch contiennent une ligne et colone dites d'initialisation.

Caractéristiques de la matrice :

- Nombre colonne = Taille(seq1)+1
- Nombre ligne = Taille(seq2)+1

		A	T	C	G	A	A	C	C	T	G
A											
T											
G											
A											
A											
T											
G											

Exercice 3 :

Créer une fonction **initmatrix** qui construit une matrice comme ci-dessus.

Java

```java
public static int[][] initMatrix(String S, String T) {
    //code de creation d'une matrice
    return matrix;
}
```

c) Initialisation la matrice :

		A	T	C	G	A	A	C	C	T	G
	0	-2									
A	-2										
T											
G											
A											
A											
T											
G											

Exerice 4 :

- Remplir la première ligne et première colonne à la main
- Créer une fonction **initglobal** qui permet de compléter la première ligne et la première colonne comme vous l'avez fait à la main ci-dessus.

Java

```java
//fonction qui ininite la premiere colonne et ligne
public static int[][] initGlobal( int[][] matrix) {
        //calcul de la premiere ligne
        //calcul de la premiere colonne
    return matrix;
    }
```

d) Remplissage la matrice :

		A	T	C	G	A	A	C	C	T	G
	0	-2									
A	-2	**3**									
T											
G											
A											
A											
T											
G											

Exerice 5 :

- Remplir la matrice à la main. Pour chaque case pointer à l'aide d'une flèche la case à partir de laquelle le calcul a été fait (voir exemple si dessus). Attention : pour remplir une case il est nécessaire de connaitre les 3 cases pouvant la précéder.
- Coder la fonction **max3** qui compare trois scores et retourne le plus élevé et qui identifiera le score maximum entre un déplacement en diagonale, vers le bas ou vers la droite.

Java

```
15  public static int max3(int a, int b, int c){
16      // comparaison entre trois valeurs et retourne la plus elevee
17      }
```

- Coder la fonction **score** qui permettra de savoir si le déplacement en diagonale est un match ou un mismatch et de retourner le score

Java

```
18  public static int score(char s,char t) {
19      // votre code de comparaison pour savoir s'il s'agit d'un match ou mismatch
20      }
```

- Coder la fonction **fillNW** qui permet de remplir la matrice et renvoi l'élément trouvé dans la case la plus en bas à droite, en utilisant : **max3** et **score**

Java

```
21  public static int fillNW(int[][] matrix, String S, String T) {
22      for (int i=1; i<=S.length();i++ ) {
23          for (int j=1; j<=T.length();j++) {
24              //calcul du meilleur score pour la case matrix[i][j]
25              }
26          }
27      return matrix[S.length()][T.length()];
28      }
```

e) Détection du meilleur alignement.

Pour déterminer le meilleur alignement, on part de la case la plus bas à droite, on suit ensuite les flèches pour remonter la matrice, tout en retranscrivant l'alignement.

Exercice 6 :

- Déterminer le meilleur alignement à la main.
- Que fait la fonction **whichmax3** ?
- A quoi correspondent a,b,c ?

Java

```
29 public static int whichMax3(int a, int b, int c) {
30     if (a>b) {
31         if (a>c) {return 1;}
32         else {return 3;}
33         }
34     else {
35         if (b>c) {return 2;}
36         else {return 3;}
37         }
38     }
```

- Compléter la fonction **GetAln**, qui va chercher et afficher le meilleur alignement global.
- Vous afficherez l'alignement selon le modèle suivant :

Sequence	match	mismatch	gap S	gap T
S	T	A	T	-
matching	|	.		
T	T	C	-	G

Java

```
39 public static void getAln(int[][] matrix, String S, String T) {
40     String alS="";
41     String alT="";
42     String matching="";
43     int i =S.length();
44     int j = T.length();
45
46     while (i>0 & j>0) {
47         int coming=whichMax3(matrix[i-1][j]+INDEL,matrix[i][j-1]+INDEL,matrix[i-1][j
       -1]+score(S.charAt(i-1),T.charAt(j-1)));
48         if (coming==3) {
49             //Actualiser als,alT et matching
50         }
51         else if (coming==1) {
52             //Actualiser als,alT et matching
53         }
54         else {
55             //Actualiser als,alT et matching
56         }
57     }
58
59     //commentaire : gap dans T si T>S
60     while (i>0) {
61         alT=T.charAt(j-1)+alT;
62         alS="-"+alS;
```

```
63          j=j-1;
64          matching=" "+matching;
65      }
66
67      //commentaire : gap dans S si S>T
68      while (j>0) {
69          alT=T.charAt(j-1)+alT;
70          alS="-"+alS;
71          j=j-1;
72          matching=" "+matching;
73      }
74      //imprimer alS, matching et alT
75  }
```

Exercice 7 :

Compléter le **main** et réaliser l'alignement de S et T.

Exercice bonus :

Coder une fonction **printMatrix** qui réalise une sortie graphique de la matrice.