TP noté – Programmation dynamique

16 avril 2019

1 Introduction

L'ARN a longtemps été considéré comme un simple messager, acteur passif de la traduction, alors que nous découvrons maintenant qu'il existe une multitude d'ARN non-codants, donc non destinés à être traduits. Ces ARN interviennent directement dans divers mécanismes moléculaires, et leurs fonctions sont extrêmement (voire uniquement) déterminée par leur structure. Connaître la structure d'un ARN étudié suscite donc un grand intérêt.

Mais les méthodes biochimiques (cristallographie, RMN...) sont coûteuses, longues et parfois incapables de donner des résultats. La prédiction de structures par des outils informatique s'impose donc comme une alternative nécessaire. Dans ce sens, énormément de méthodes ont été développées. Nous allons nous intéresser à l'une d'entre elles, dont le concept de base est conservé dans la plupart des algorithmes de prédiction *ab initio* actuels.

2 L'algorithme de Nussinov

Les nucléotides d'une molécule d'ARN interagissent entre eux, créant des liaisons de différents types. Parmi elles, se trouvent les *Watson-Crick* et les *Wooble* dont on considère qu'elles constituent le squelette de la structure.

La première (et importante) étape dans la prédiction de structure d'ARN est, bien souvent, de déterminer ce que l'on nomme la structure secondaire, c'est-à-dire l'ensemble des liaisons *Watson-Crick* et *Wobble* avec une contrainte supplémentaire :

> La structure est sans pseudo-nœud, c'est-à-dire que les liaisons ne se croisent pas.
> *Ainsi, si un nucléotide en position i possède une liaison avec un nucléotide en position j ($i < j$), alors tout nucléotide placé entre i et j ne peut faire de liaison qu'avec un autre nucléotide lui aussi placé entre i et j.*

Cette contrainte est fondamentale pour pouvoir appliquer l'algorithme de Nussinov.

L'algorithme de Nussinov consiste à déterminer la structure secondaire d'un ARN donné, **telle qu'elle maximise le nombre de liaisons**. L'algorithme repose sur le principe de **la programmation dynamique** : il calcule la meilleure structure pour des petites sous-séquences et l'utilise pour en déduire des structures de séquences de plus en plus grandes.

Ainsi en définissant :
- S la séquence en nucléotides de l'ARN (de longueur N et numérotée de 0 à $N-1$),
- $S(i,j)$ la séquence entre les nucléotides i et j (indicée de 5' à 3'),

l'idée clé de ce calcul par récurrence est qu'il n'existe que quatre possibilités pour obtenir la meilleure structure de $S(i,j)$ (voir la figure 1) :

1. ajouter une liaison entre i et j sur la meilleure structure de $S(i+1, j-1)$.
2. ne pas ajouter de liaison à i sur la meilleure structure de $S(i+1, j)$.
3. ne pas ajouter de liaison à j sur la meilleure structure de $S(i, j-1)$.
4. combiner deux sous-structures optimales $S(i,k)$ et $S(k+1,j)$ ($i < k < j$).

Plus formellement,
- soit x_i le nucléotide à la position i,
- $\delta(i,j)$ la fonction telle que : $\delta(i,j) = 1$ si x_i et x_j peuvent former une liaison Watson-Crick ou Wooble, et $\delta(i,j) = 0$ sinon.

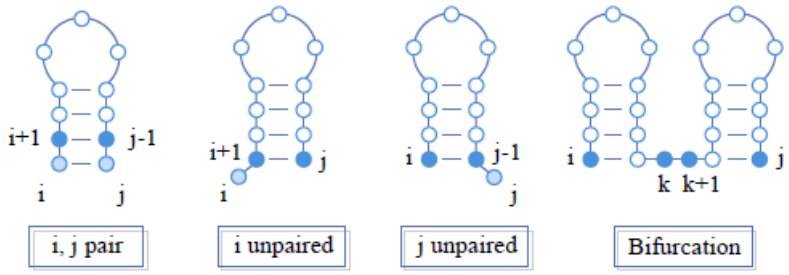

FIGURE 1 – *L'algorithme de Nussinov s'interesse à quatre possibilités pour trouver la meilleure structure de la sous-séquence $S(i,j)$ par ajout de i et/ou j sur une structure optimale déjà calculée d'une sous-séquence plus petite.*

On calcule alors le score $\gamma(i,j)$ (représentant le nombre maximum de liaisons que l'on peut réaliser pour une séquence donnée) par la formule de Nussinov-Jacobson :

$$\gamma(i,j) = max \begin{cases} \gamma(i+1,j-1) + \delta(i,j) \text{ (cas 1)} \\ max_{i \leq k < j}[\gamma(i,k) + \gamma(k+1,j)] \text{ (cas 2)} \end{cases}$$

avec les valeurs initiales suivantes :

- $\gamma(i,i) = 0$ pour $0 \leq i \leq N-1$
  En effet, un nucléotide ne peut pas s'apparier avec lui-même...
- $\gamma(i,i-1) = 0$ pour $1 \leq i \leq N-1$
  Deux nucléotides voisins ne s'apparient pas.

Le cas 1 correspond à la première possibilité, et le cas 2 regroupe les 3 autres.

3 Le programme

3.1 L'entrée du programme

Le programme que vous devez réaliser doit prédire la structure secondaire (par la méthode de Nussinov) d'un ARN dont la séquence est contenue dans une chaîne de caractères de taille N.

3.2 La sortie

Le programme doit retourner la structure secondaire sous forme de parenthésages, alignée avec la séquence comme dans l'exemple suivant :

A	U	G	C	C	U	G	G	C	U	A	G	U	U
(	.	(	(	(	.	)	)	)	(	.	)	.	)

3.3 L'implémentation

Vous utiliserez deux matrices M_S et M_D pour stocker les scores et les directions respectivement.

Les étapes :

1. En premier lieu, créez une fonction `int delta(char c1, char c2)` qui renvoie 1 pour pour les liaisons C-G, G-C, A-U, U-A, U-G, G-U, et 0 sinon.
2. Créez une fonction `void fctGamma(int** ms, int** md, int i, int j, char* seq)` qui calcule le score γ :
   — $cas1 = \gamma(i+1,j-1)$ quand i et j sont des nucléotides voisins,
   $cas1 = \gamma(i+1,j-1) + \delta(i,j)$ sinon.

— $cas2 = max_{i \leq k < j}[\gamma(i,k) + \gamma(k+1,j)]$) *(Pensez à faire une boucle pour trouver ce maximum...)*
— si $cas1 \geq cas2$, $\gamma(i,j) = cas1$, sinon $\gamma(i,j) = cas2$

Le score γ calculé est stocké dans la case (i,j) du tableau M_S, qui est une matrice carrée de la taille de la longueur de la séquence.

Parallèlement, on stocke dans la matrice M_D (matrice de même taille que M_S) :
— la valeur $-\delta(i,j) - 1$ si on a utilisé le cas 1,
— la valeur k si on a utilisé le cas 2. *(Dans la boucle du cas 2, pensez à stocker un $kMax$ pour savoir quel était le k pour lequel la valeur γ était maximale...afin de pouvoir stocker ce $kMax$ dans M_D)*

Aide :
Avant l'étape 2, pensez à créer les fonctions :

- `int** allouеMatrice(int n)` afin de pouvoir l'appeler depuis votre main de cette façon : `int** ms = allouеMatrice(n);` et `int** md = allouеMatrice(n);`
- `void freeMatrice(int** M,int n)`
- `void afficheMatrice(int** M, int n)`

Résultat : **La valeur de $\gamma(0, N-1)$ est le nombre maximum de liaisons que l'on peut réaliser, et la structure recherchée s'obtient par remontée dans la matrice M_D.**
Pour cela, créer la fonction récursive `trouverStruct(int** md, int n, char* structure, int i, int j)`.

- La chaîne de caractère *structure* stockera les parenthèses et les points représentant la structure.
- La condition d'arrêt de la récursivité sera soit $i > j$,
  soit $i == j$ où il faudra stocker un point à la case i de la chaîne *structure*.
- **Cas 1 :**
  Il vous faudra considérer le cas où la valeur -2 est stockée dans la matrice M_D,
  ce qui signifie qu'il y a appariement (dans ce cas $structure[i]$ ='(' et $structure[j]$ =')').
  Il vous faudra considérer le cas où la valeur -1 est stockée dans la matrice M_D,
  ce qui signifie que l'appariement est impossible, mais que le cas 1 est tout de même préférable au cas 2 (dans ce cas $structure[i]$ ='.' et $structure[j]$ ='.').
  Pour -1 comme pour -2, après avoir stocké les points ou parenthèses dans la chaîne *structure*, vous appelez récursivement `trouverStruct` avec les indice $i+1$ et $j-1$ puisqu'on s'est déplacé en diagonale.
- **Cas 2 :**
  Si $M_D[i][j]$ est différent de -1 et de -2, alors vous étiez dans le cas 2, et c'est la valeur de k qui est dans la matrice. Vous n'avez donc dans cette condition qu'à appeler la fonction `trouverStruct` avec :
  — (i,k)
  — $(k+1,j)$

Depuis votre `main`, vous appelez `fctGamma` "diagonale par diagonale" (vive les boucles !).