But du projet : réussir à programmer un algorithme de prédiction de structure des ARN

Documents utilisés :

**Thèse d’Olivier Perriquet “Approche algorithmique pour la prédiction de la structure secondaire des ARN “**

disponible sur

<https://ori-nuxeo.univ-lille1.fr/nuxeo/site/esupversions/f737526f-4ef5-4eaa-bb67-669a8e3559ed>

Etape 1 : Comprendre, s’approprier et implémenter l’algorithme de Nussinov et Jacobson

Cette méthode est une approche thermodynamique du problème que l’on cherche à résoudre.

C’est une méthode de programmation dynamique.

Prenons une séquence d’ARN de taille n.

Pour pouvoir appliquer l’algorithme de Nussinov et Jacobson, il faut remplir une matrice de taille n².

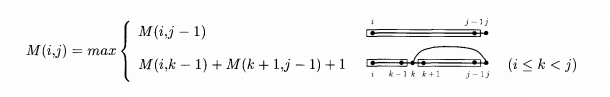
L’algorithme se décompose en deux étapes.

Tout d’abord il faut remplir la matrice.

Celle-ci contient dans chaque case le maximum d’appariements réalisables

On sait que A correspond à T et G correspond à C.

Formellement, cela s’écrit :



On commence par remplir la 1ère diagonale avec des 0 car une base ne peut pas être appariée avec elle-même (A ne correspond pas à A, ni G avec G, C avec C ou T avec T).

On ne se sert que de la moitié de la matrice pour ne pas avoir à écrire 2 fois les mêmes données.

Ensuite pour chaque case, on calcule si elle reste libre ou si elle peut être appariée avec une autre.

Dans tous les cas, on prendra la plus grande valeur possible.

On peut calculer cette valeur de 2 manières différentes, donnant 2 résultats qui peuvent donc être différents.

Si la base est libre, alors on prend la valeur de la case se trouvant en dessous.

Sinon, si la base j forme un appariement avec la base k, on ajoute 1 à la somme des valeurs déjà présentes dans la matrice en [i] [k-1] (la valeur de la case juste avant la position de la base k) et en [k+1][j-1] (la valeur de la case en diagonale de la case [j][k] dont on essaye de déterminer la valeur).

2ème étape :

Le rebroussement

Le score maximal d’appariements qui peuvent être formés à partir de la séquence donnée en entrée se trouve dans la case [0][n-1] (en haut à droite) puisque c’est la dernière case dont on va calculer la valeur.

Comme pour un algorithme d’alignement de séquence ADN de Needleman et Wunch, il faut rebrousser le “chemin” d’appariements de ribonucléotides qui nous ont permis d’obtenir ce score. Ainsi nous aurons tous les appariements que nous avons calculés auparavant.

Pour cela, il faut retrouver comment nous sommes parvenus à ce score.

Partie Pseudo-code

typedef struct {

char base;

int position;

int eng;

int position\_lie;

} APP;

APP tab [strlen(seq)][strlen(seq)] ;

void remplir\_matrice (){

int energie = 0;

for (int i = 0; i<strlen(seq); ++i){

tab[i][i].base = seq[i];

tab[i][i].position = i;

tab[i][i].eng = 0;

tab[i][i].position\_lie = -1;

}

for(i = 0; i < strlen(seq);++i){

j = ++i;

tab[i][j].base = seq[j];

tab[i][j].position = j;

tab[i][j].eng = tab[i][j-1]

tab[i][j].position\_lie = -1;

for (int k = min(i, j); k < max (i, j); ++k){

if(seq[j] == “A” && seq[k] == “T” || seq[j] == “T” && seq[k] == “A” || seq[j] == “G” && seq[k] == “C” || seq[j] == “C” && seq[k] == “G”) {

energie = T[i][k-1].eng + T[k+1][j-1].eng+1;

if (energie > T[i][j].eng){

T[i][j].eng = energie;

T[i][j].position\_lie =k;

T[i][k].position\_lie = j;

}

}

}

}

}

void rebroussement(){

int i, j;

char \*appariement;

i = 0;

j = strlen(seq)-1;

while (tab[i][j].eng != 0){

if (tab[i][j].position\_lié != -1){

if (tab[i][j].position\_lie < tab[i][j].position) appariement = “)” + appariement;

else appariement = “(“ + appariement;

i = ++i;

j = --j;

}

else { appariement = “-” + appariement;

j = --j;

}

}

printf (“%s \n”, appariement);

}

Partie II – Algorithme de Zucker

Comme expliqué dans la thèse, on ne veut plus maximiser le nombre d’appariements possibles mais se rapprocher au mieux de la réalité biologique en optimisant l’énergie libre.

On apprend également qu’une molécule est faite de « modules » ayant une énergie libre propre.

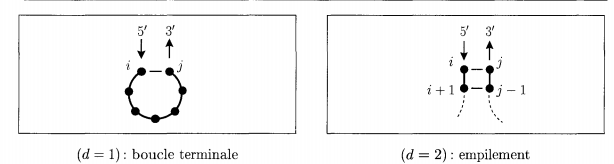
Cette énergie libre est négative, puisqu’on recherche le repliement « naturel » de la molécule d’ARN, la forme optimale qu’il prend sans apport d’énergie nécessaire.

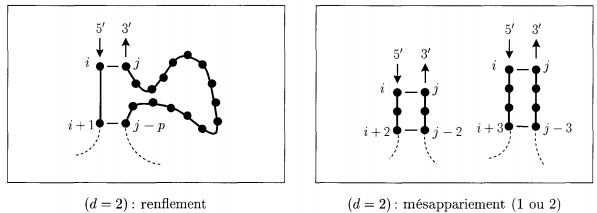
L’énergie libre de la molécule est logiquement la somme des énergies libres des modules qui la composent.

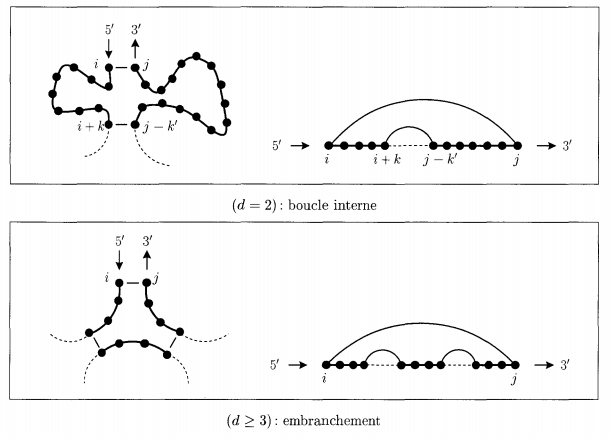
Les modules ont par ailleurs un degré que l’on peut calculer à partir du nombre d’appariements réalisés par chacune des bases de l’appariement « fermant ».

Ce degré permet de quantifier, de manipuler et de traduire un phénomène biologique (formation d’une boucle, d’un renflement ou d’une tige) dans un algorithme, rendant celui-ci bien plus proche de la réalité.

L’algorithme de Zucker s’appuie sur ces degrés de modules.







Classification des modules selon le degré (tirée de la thèse d’Olivier Perriquet)

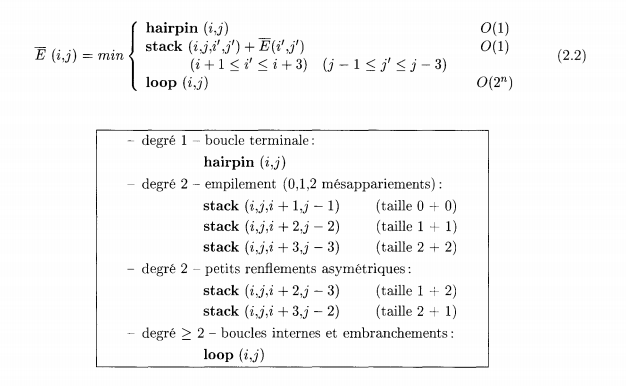
L’algorithme comme pour Nussinov Jacobson utilise la programmation dynamique.

On se sert d’une matrice E de taille n² (n étant la taille de la séquence) et d’une matrice auxiliaire de même taille qu’on appellera E’.

Entre deux bases i et j, E[i][j] contient l’énergie libre de la molécule seq[i…j] (donc l’énergie libre de tous les modules contenus entre i et j).

Entre deux bases i et j, E’[i][j] contient l’énergie libre de la molécule seq[i…j] **dans l’hypothèse d’un appariement entre i et j**.

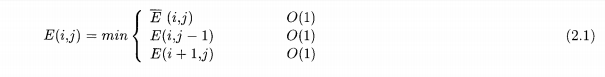
La matrice auxiliaire E’[i][j] est calculée en prenant le minimum des 3 valeurs suivantes pour chaque appariement E’[i][j] :



Calcul de la valeur de E’[i][j]

On se sert ici donc de la valeur du degré de chaque module en prenant l’appariement en compte : si on a un appariement, quel sera le module avec l’énergie libre la plus basse (donc le plus facilement créé lors du repliement optimal de l’ARN).

On se sert ensuite de cette valeur pour calculer E[i][j].



(ici E’ = 6.PNG)

On calcule ensuite la valeur minimum entre E’[i][j] et E[i][j-1] et E[i+1,j] (on compare en réalité les valeurs d’énergie libre entre une molécule qui possède un appariement entre i et j et une molécule qui ne possède pas un tel appariement et qui donc a l’énergie libre « transmise » par le reste de la molécule, en [i][j-1] ou en [i+1][j]).

Ainsi, E[i][j] <= E’[i][j] (étant donné qu’on prend le minimum).