Compte-rendu TP2 AeA

Claire Hunter Antoine Canda

13 mars 2017

1 Introduction

Ce projet a pour objectif de nous faire implémenter un outil permettant la détection de microARNs qui présentent des spécificités propres comme l'appartenance à un précurseur qui présente une structure "tige-boucle" avec un appariement relativement important sans être parfait (d'où les boucles).

2 Écriture des tests

Nous avons écrit quelques fichiers de tests présentant différentes spécificités grâce à l'outil du site bioinformatics. On a par exemple des cas présentant uniquement des appariements forts, puis on inclut petit à petit des boucles terminales, internes, des appariements GU. Enfin, nous entourons de séquences aléatoires la structure tige-boucle. Ces fichiers sont présents sous le nom fileTestX.fasta dans le dossier fichier-test.

3 Algorithme de détection

Nous avons tenté de mettre en place un algorithme de programmation dynamique permettant la détection des structures tiges-boucles mais nous n'avons pas eu le temps de le finaliser et il n'est malheureusement pas fonctionnel en l'état.

Pour détecter les structures, nous prenons des séquences d'ADN de taille 100 qui constituerait la fenêtre que l'on étudie à un instant t. On déplacerait cette fenêtre de 1 nucléotide si on ne trouve pas de structure candidate ou de 100 dans le cas où on en trouve une.

Pour mettre en place une stratégie de programmation dynamique nous commençons par créer une matrice de taille 50 sur 50 avec l'idée suivante : on considère que les colonnes correspondent aux nucléotides des positions 0 à 49 et que les lignes correspondent aux nucléotides des positions 99 à 50 de notre séquence. Le départ de notre table est en haut à gauche avec les nucléotides (0,99) que l'on initialise avec la valeur "U" pour undefined.

L'idée est de parcourir la table au maximum par la diagonale afin d'éviter des traitements inutiles que l'on ferait si on décidait de remplir toute la table.

Dans l'algorithme complet, la table dispose de trois zones en quelque sorte. La première va des colonnes 0 à 14 et des lignes 99 à 85. Cette zone est une sorte de zone tampon dans le cas où on a une structure de taille comprise entre 70 et 100 exclu et elle peut avoir des séquences ne respectant pas les critères de définition des structures que l'on a à savoir une suite de 3 appariements successifs maximum ainsi que des boucles internes de taille supérieur à 3 nucléotides qui correspondraient simplement à une séquence d'ADN ne faisant pas partie de la structure recherchée. L'implémentation de notre algorithme ne comprend pas cette zone.

Une seconde zone se trouverait aux colonnes 46 à 49 et aux lignes 53 à 50. Elle représente la partie où se trouve la boucle terminale s'il y en a une et on y autorise donc jusqu'à 4 mismatchs (ce qui correspondrait à 8 points maximum car ce sont deux parties contiguës).

Enfin, la troisième zone se trouve entre les deux autres et c'est là où se trouve, quelque soit la longueur de la structure tige-boucle, la partie de séquence d'ADN (ou ARN) devant respecter scrupuleusement les règles concernant les groupes d'appariements ou de non-appariements.

En respectant cette définition nous avons détecté un cas limite qui correspondrait au cas où on a un micro ARN qui se trouve entre les positions 0 et 100 avec une taille inférieur à 100 nucléotides. Une solution serait de rendre plus générique notre algorithme sur la taille et de lancer pour la première fenêtre 15 résolutions avec des chaînes de taille 70 à 100 qui augmentent de 2 nucléotides à chaque occurrence. Cette solution aura un impact d'autant plus faible si la

séquence d'ADN traitée est grande et inversement d'autant plus forte si la séquence traitée est petite. Toutefois, elle permet d'être exhaustive et avec une heuristique on pourrait arrêter cette phase et passer directement à l'utilisation de fenêtres de taille 100 après la fin de la structure détectée si on en détecte.

Maintenant que l'on a défini les différentes zones de notre table et comment l'utiliser sur une séquence d'ADN de taille n, nous allons nous intéresser au coeur de l'algorithme qui est le remplissage de la table. On a également crée un tableau chemin permettant d'effectuer un backtracking de notre solution.

L'idée de base est de considérer au maximum la diagonale de notre table. On va donc considérer s'il y a un appariement entre les nucléotides en position i et j. Si c'est le cas et que c'est la première fois que l'on considère cette appariement alors on met la valeur "M" dans la case [i][j] et on passe à la diagonale de cette case. C'est le cas simple et idéal du parcours.

S'il n'y a pas d'appariement alors on marque la case de la valeur "X" qui a trois significations relativement proches : soit on est en présence d'un mismatch si on continue le parcours par la diagonale, soit on est en présence d'une insertion si on continue par une descente vers le bas (ligne) ou d'une délétion si on poursuit le parcours vers la droite (colonne). Dans la suite, on appellera un "non-match" tout ce qui est soit un mismatch, soit une insertion, soit une délétion.

Il nous a fallu gérer le cas où on tombe sur plusieurs non-matchs de façon symétrique, c'est à dire par exemple le cas où on a (((**...**(((...)))**...**))). On a alors considéré que les non-matchs peuvent être acceptés dans le cadre d'un carré de taille 3 sur 3 avec donc un maximum de 5 à la suite ce qui correspondrait au cas où on a 3 non-matchs de chaque côté qui sont présents au même moment.

On a eu besoin également de créer une fonction de backtrack pour remonter dans la table et en cas d'appariements impossibles on passe les valeurs des cases à "I" pour invalide. L'arrêt du parcours peut notamment se faire si on a pas de solution suite au backtrack ou que l'on a plus de 24 appariements valides.

On n'a pas eu le temps de finaliser cette partie avec notamment la gestion de la boucle terminale.

Quand on a un chemin non vide, c'est à dire quand on a détecté une structure candidate, on a essayé de faire une remontée de table pour recréer la structure sous forme (((.(((.. ... .))))..))).
Usage : java -jar aea.jar <fichier.fasta> mais non fonctionnel dans l'état.

4 Analyse des résultats

Nous n'avons pas eu le temps de faire cette partie du TP souhaitant passer plus de temps sur la partie algorithmique de la phase 2.

5 Conclusion

Malgré que l'on n'ait pas eu le temps de finir ce TP, nous avons cherché une solution un minimum optimale notamment sur papier et Excel avec un travail de simulation de déroulement de l'algorithme sur l'exemple du poly et ensuite on a tenté de l'implémenter mais le manque de temps nous a un peu posé problème.