Université de Sherbrooke

Faculté des sciences

Département d’informatique

Baccalauréat en informatique

Projet Motif Planté

Par

William O’Sullivan-Dargis, osuw3401

Travail présenté à Mr Manuel Lafond

Dans le cadre du cours

BIN702 - Algorithmes pour la bioinformatique

X x 2021

## Introduction

## Revue de littérature

<https://arxiv.org/abs/1307.0571>

**Sample driven part**

On génère tout les l-mers des S dans une matrice R tel que la colonne Si contient ses l-mers.

On prend un l-mer x de S1 et on filtre dans R tous les l-mers qui sont à une distance plus grande que 2d de x, car s’ils sont à une distance plus grande que 2d, alors ils n’ont nécessairement pas de voisin commun. Ensuite, on ajoute un l-mer de S2 dans R sur la pile et on filtre tous les l-mers dans R qui n’ont pas de voisin commun avec les l-mers sur la pile. Et ainsi de suite.

Si un S (une colonne de R) n’a plus de l-mer, alors on dépile le dernier l-mer x de Si, on réinitialise R à son état précédant et on choisis un autre l-mer de Si.

**Pattern driven part**

Si la pile est assez grande, alors on génère le voisinage commun entre les l-mer de la pile.

Pour chaque voisin généré, on vérifie si chaque colonne de R contient au moin un l-mer à une distance de hamming plus petite ou égal à d du voisin. Si c’est le cas, alors le voisin est un motif.

Prend O((n\*m)2) espace

## Méthodologie

Hill climbing :

Marche très bien et est très rapide. Par contre son taux de succès dépend fortement sur le nombre de séquence, la longueur des séquences ainsi que la distance entre le motif et l’instance de celui dans chacune des séquences.

Lorsqu’on génères les séquences de tests, on va générer un motif et implanter une instance de celui qui est à distance <= d dans chacune des séquences. Si la distance entre les instances et le motif est d’exactement 12 pour toute les instances, alors avec l = 26 et d = 11, l’algorithme de hill climbing aurra un taux de succès très bas. Si la distance entre les instances et le motif est généré au hasard avec une probabilité égal pour d = 1..11, alors le taux de succès est de 99%.

Complexité :

Pour chaque l-mer (seqLen),

on génère les voisins à distance 1 ( l \* |A|),

on calcule les hamming distances avec tout les autres l-mers de tout les autre séquences (l \* nbrSeq),

on climb au maximum d fois, car si on change > d lettres, alors c’est pas un motif (d). (So so vrai en pratique, ca impacte pt la precision de faire ca, car on peut changer plus de d et revenir après).

O(l2 \* d \* |A| \* seqLen \* nbrSeq)

O(l2 \* |A| \* seqLen \* nbrSeq \* nbrIterationClimb)

Le principe est le suivant, on génère les l-mers d’une séquence arbitraire et on évalue leurs distances comme suis :

Le principe général du hill climbing est le suivant : on part d’une séquence arbitraire X de longueur l, on explore les voisins X’ à une distance k de X. On sélectionne le voisin qui maximise un score (ou qui le minimise dans notre cas) et on recommence jusqu’à temps d’atteindre un maxima local ou une chaine qui est à une distance de Hamming plus petite ou égal à d d’un l-mers dans tous les séquences.

**Candidat débutant la recherche**

Puisqu’on sait que si une solution existe, alors chaque séquence contient nécessairement un l-mers à distance plus petite ou égal à d, alors on peut débuter la recherche à ses l-mers. On va donc faire une recherche pour chaque l-mers d’une séquence arbitraire.

**Exploration des voisins**

Un candidat de longueur l à C(l, d) \* (|A|-1)d voisins à distance plus petite ou égal à d. Donc, le nombre de voisins à explorer augment exponentiellement avec la distance. C’est pourquoi nous allons d’abord explorer les voisins à distance de 0 et si aucun motif n’est trouvé on va alors continuer avec une distance de 1, puis 2, etc. Augmenter la distance de recherche permet de potentiellement sortir du minimum local.

Une optimisation importante pour l’exploration des voisins est de trier les résultats des explorations. Ceci fait en sorte que nous explorons les voisins plus probables d’être le motif en premier.

De plus, étant donné que les recherches des voisins à une distance plus grande que 1 peut être assez longue, on peut éliminer un certain pourcentage des candidats pour seulement évaluer les candidat les plus probables.

TODO : Ca marche parce que à la place d’explorer tout les voisins à distances d, on explore les voisins près et on se rapproche tranquillement du voisin à dist d. Donc on explore une petite zone à l’entour du chemin au lieu d’explorer un gigantesque zone.

**Évaluation des distances**

Pour évaluer les distances, il est intuitif de prendre la formule suivante :

Elle décrit la distance maximale du candidat X au l-mers le plus près de chaque séquence. Si cette valeur est plus petite ou égal à d, alors on a trouvé notre motif. Il ferait donc du sens de prendre cette distance pour faire le hill climbing, car c’est ultimement ceci que l’on veut minimiser. Cependant, en pratique, cette valeur ne varie pas beaucoup entre les candidats et leurs voisins. Alors, dans la plupart des cas, aucun voisin n’aura une meilleure valeur que le candidat courant. Donc, il est difficile de s’approcher d’une solution celle-ci, car si aucun voisin n’a une meilleure valeur on ne sait pas dans quelle direction explorer.

Pour résoudre ce problème et être capable de continuer de s’approcher de la solution, on gère les cas d’égalité avec la distance suivante :

Le fait que la distance soit au carré permet de favoriser les candidats qui sont à des distances équilibré des séquences. Par exemple, si on a les candidats avec les distances suivante (1,12,12) et (6,7,12), bien que pour les 2 la distance maximale est 12 et la somme des distances est 25, le deuxième candidat est probablement meilleur, car on doit seulement réduire une seule distance. Cela fonctionne, car tant qu’une distance est plus basse que d, alors celle-ci est acceptable.

## Résultats

## Conclusion

## Description et objectifs

Bonjour, pour mon projet, je vais travailler sur le problème du motif planté.

Le problème est décris au point 8 des propositions de projet « Dans ce problème, on a n séquences s1, . . . , sn et deux entiers l et d. On doit trouver une séquence X telle que |X| = l et telle que chaque chaîne si contient une 2 sous-chaîne X0 qui est à d modifications près de X »

Je souhaite étudier et comprendre certain des algorithmes communs exacts et d’approximations ainsi que développer et implémenter un algorithme exact et/ou d’approximation et comparer sont temps d’exécution théorique et pratique aux autres algorithmes commun. À priori, développer un heuristique pour un algorithme d’approximation rapide m’intéresse un peu plus, mais j’aimerais aussi explorer les algorithmes exacts.

Par exemple, j’aimerais le tester sur 20 chaines de 600 charactères pigés au hasard dans un alphabet 4 caractères avec un l = 26 et d = 11.