Algorithmique et Programmation Avancées pour les Biologistes

# Recherche de motifs

### Motivation

Facteurs de transcription

Tous les algos qui font de la découverte de motif en bio, ils sont validés avec l’identification de sites de fixation de facteurs de transcription, et d’une façon générale, tout ce qui est site fonctionnel (ils sont donc conservés).

### Détection de motifs versus recherche de motif.

Motif connu (ou un type de motif, style regexp : un motif avec des gaps est un motif structurel) -> on le cherche dans une séquence. Ça c’est de la détection. Un type de description de motif détaillé est un pssm. Des occurrences de motifs (dans un jeu de données d’apprentissage) permettent de construire un modèle de motifs. Ce modèle s’appelle la pssm : Position Specific Score Matrix.

Occurrences d’un motif dans un jeu de séquences

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| % | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| a | 40 | 5 |  |  |  |  |  |
| c | 20 | 5 |  |  |  |  |  |
| t | 20 | 40 |  |  |  |  |  |
| g | 20 | 50 |  |  |  |  |  |

PSSM

Ici, la détection est beaucoup plus souple.

Dans la recherche de motif, en général, le motif est complètement inconnu. Ici, on a n séquences comportant chacune une occurrence du motif inconnu : quorum à 100%

: Ce motif n’est pas forcément exactement le même dans chaque séquence.

Quorum à 40% : existe-t-il au moins 40% des n séquences comportant chacune une occurrence du motif ?

* L’occurrence exacte d’un motif : Dans une séquence, c’est un fragment qui est exactement identique au motif recherché.
* Occurrence approchée d’un motif :
  + Avec erreur de substitution ( = mismatch) : motif et occurrence ont la même longueur. La référence est le motif recherché = le modèle.

ACTGG -> *T*CT*A*G

* + Avec erreur d’insertion et de délétion : motif et occurrence ont des longueurs différentes.

ACTGG -> AC*A*TGG : 1 erreur d’insertion.

ACTGG -> AGG : 2 erreurs de substitution.

**Algorithme exact** : donne vraiment la solution exacte, mais met des heures à s’exécuter. Aucun compromis, il respecte exactement les critères énoncés dans le problème. La solution est optimale. Ne convient pas aux problèmes de grande taille (en général).

**Algorithme approché** : fournit une/des solution(s) approchée(s), sous-optimale. C’est un algorithme heuristique. La qualité doit quand même être suffisante. Se fait au prix d’un compromis entre durée d’exécution et qualité de la/des solution(s).

### Résolution du problème de découverte de motif (inconnu) commun à plusieurs séquences par un algorithme exact.

3 versions : - occurrences exactes

- occurrences approchées, avec erreurs de substitutions seulement.

- occurrences approchées, cas général (substitution, insertion, deletion).

Avec occurrences exactes.

Conventions :

Motif appelé ici modèle = M (actgg)

actgg

: Occurrence ( )

Si M = « actgg », et qu’on y ajoute un caractère voisin direct ‘a’, et que cela forme le nouveau modèle, celui-ci peut s’appeler le modèle ‘Ma’.

Dictionnaire de modèles de longueur i = 4.

Aaaa

Aaac

Aaat

…

Tttt

 Ce sont tous les modèles que l’on peut extraire d’un jeu de données. Avec un jeu de séquences restreint, on aura pas forcément tous les modèles possibles (selon la combinatoire).

Puis on crée le dictionnaire des modèles de longueur i = 5, pour le même jeu de séquences.

Quand on liste les modèles potentiellement présents, on ne peut pas l’assurer comme modèle en tant que tel, car on n’a pas encore examiné les occurrences des séquences suivantes.

Un algorithme de programmation dynamique est :

1. un algorithme glouton, c’est-à-dire qui procède en construisant une solution à l’itération i en étendant une solution partielle obtenue à l’itération i-1, et ceci sans retour arrière.
2. Qui assure le caractère optimal de la solution finale grâce à des propriétés assurant l’optimalité de l’extension.

Note : Retour arrière : on ne remet jamais en question la solution courante.

Complexité d’un programme : temps de calcul, nombres de calculs, d’instructions, des paramètres…

complexité n k m² : le temps d’execution est proportionnel à ça : une classe de problème avec m = 2, ça va, mais si m = 10 000, la complexité de l’algorithme est trop élevée.

Algorithme exact

Entrées : n séquences.

l longueur maximale de motif.

Identifier les modèles de longueur 1 et leurs occurrences. //construction du 1er dictionnaire.

Pour i allant de 2 à l

Pour chaque modèle M de longueur i-1

Pour chaque occurrence occ = (n° séquence, position) de ce modèle

S’il existe un caractère x voisin droit de occ //extension possible à droite

Identifier le caractère x.

Si le modèle Mx est absent du dictionnaire i

Alors Ajouter Mx au dictionnaire i.

Fin si

Ajouter occ\_nouv = (n°seq, position) aux occurrences du modèle Mx.

Fin si

Fin pour

Détruire les modèles Mx non-valides.

Fin pour

// le dictionnaire i est créé.

Si le dictionnaire i est vide

Afficher « pas de motif reconnu ».

Sortir de l’algorithme.

Fin si

Détruire le dictionnaire i-1.

Fin pour

//postcondition :

// Le dictionnaire l existe

Afficher dictionnaire l.

La version avec erreurs de substitution

Du dictionnaire M, avec un g derrière, on peut passer à l'étape i+1, avec les dictionnaires :

* dictionnaire Mg : pas d'erreur de substitution
* dictionnaire Ma : 1 erreur de substitution.
* dictionnaire Mt : 1 erreur de substitution.
* dictionnaire Mc : 1 erreur de substitution.

Si on veut pas que ça explose la mémoire (et pour que le programme reste cohérent), il faut limiter le nombre maximal de substitutions autorisées.

Algorithme avec erreurs de substitution

paramètres d'entrées : n séquences

l longueur du motif

k nombre maximum d'erreurs de substitutions autorisées.

Construire le dictionnaire 1 (i.e. Des modèles de longueur 1)

s : nombre d'erreurs de substitutions enregistrées jusqu'à présent

pour i allant de 2 à l

pour chaque modèle M du dictionnaire i-1

pour chaque occurrence occ = (N°seq, pos, s)

Si il n'existe pas de caractère x voisin droit de occ Alors continuer boucle finsi

identifier x ;

*//extension sans erreur.*

Si Mx n'existe pas dans le dictionnaire i, alors ajouter Mx au dictionnaire i. finsi

Ajouter la nouvelle occurrence occ\_nouv = (N°Seq, pos, s) aux occurrences de Mx ;

//(il vaut mieux l'ajouter en tete de liste, c'est bcp plus court).

Si (s == k) alors continuer boucle finsi

*//extensions avec 1 erreur de substitution :*

pour y!=x, y appartenant à l'alphabet sur lequel sont construites les séquences //{a,t,g,c} pour l'ADN

Si My n'existe pas dans le dictionnaire i alors ajouter Ny au dictionnaire i. finsi

Ajouter la nouvelle occurrence occ\_nouv = (N°seq, pos, s+1) aux occurrences de My.

finpour

Finpour

finpour

Eliminer du dictionnaire i tous les modeles ne respectant pas la contrainte de quorum.

Si le nouveau dictionnaire i est vide, Alors sortir de l'algorithme. Finsi

Détruire le dictionnaire i -1

Finpour

Afficher les modèles et leurs occurrences.

Soit on teste modele par modele, soit on teste à la fin. La complexité est la même. A chaque fois qu'un nouveau dictionnaire est créé, on regarde si les modeles qu'il contient respectent le quorum. Ou alors, on regarde dans le dictionnaire final.

Insertion suivi d'une deletion : revient à une substitution.

Occurrence exacte : codage : occ = (N°seq, pos)

Occurrence approchée avec des erreurs de substitution. Occ = (N°seq, pos, s)

Occurence approchée avec des erreurs de substitution, d'insertion et de délétion. Occ=(N°seq, pos, s, i, d, last)

Quand on est en train de créer un nouveau couple(modèle, occurrence) « 2e opération », on a besoin de savoir quelle était la nature de l'opération élémentaire précédente : c'est mémorisé en last.

Après analyse de ces redondances, on s'aperçoit de la ppté suivante :

Un insertion ne peut que suivre un match ou une autre insertion.

On ne veut pas voir une insertion qui suit une substitution.

Algorithme avec erreurs de substitution et délétions

paramètres d'entrée : n séquences

l longueur du motif

k nombre maximum d'erreurs autorisé.

match\_possible : booléen

creer\_le\_dictionnaire\_initial //toutes les occurrences du modèle vide.

Pour it allant de 1 à l :

Pour chaque modèle M :

Pour chaque occurrence occ=(N°seq, pos, s, i, d, last) :

j = pos + i – d + it

traiter\_substitution(M, occ, k, Data[N°Seq], j)

match\_possible ← vrai

traiter\_match(M, occ, k, Data[N°Seq], match\_possible, j)

traiter\_deletion(M, occ, k, Data[N°Seq])

traiter\_insertion(M, occ, k, Data[N°Seq], match\_possible, j)

FinPour

FinPour

Supprimer\_modeles\_ne\_verifiant\_pas\_quorum\_dans\_dictionnaire\_en\_cours\_de\_construction( 100 % par defaut)

Si dictionnaire en cours de construction est vide Alors sortir algo FinSi

detruire\_dictionnaire(i-1)

FinPour

PROCÉDURE traiter\_substitution(ȸM, ȸ occ=(N°seq, pos, s, i, d, last),ȸ k, ȸ S, ȸ j)

//S est la séquence dans laquelle est localisée occ.

Switch(last){

subst :

Pour tout x!= S[j] :

Si Hx n'est pas déjà dans le dictionnaire alors le créer là

Ajouter occ\_new = (N°seq, pos, s+1, i, d, subst) à la liste des occurrences de Mx.

FinPour

insert :

match :

del :

rien à faire

}