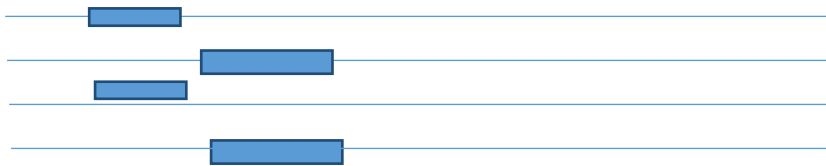


Algorithmique et Programmation Avancées pour les Biologistes

Recherche de motifs

Motivation

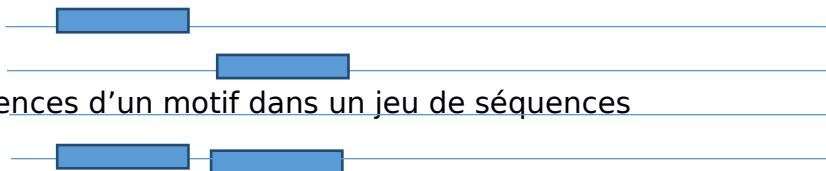
Facteurs de transcription



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 structural) -> 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.



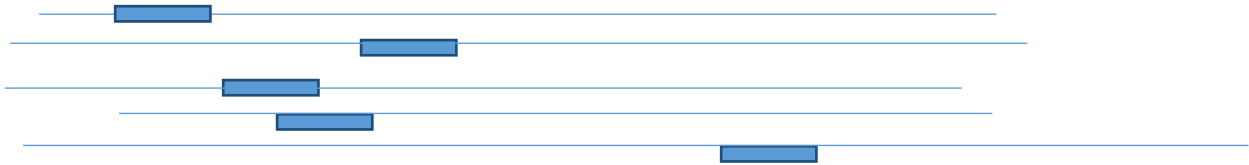
Occurrences d'un motif dans un jeu de séquences


%	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 -> TCTAG
 - Avec erreur d'insertion et de délétion : motif et occurrence ont des longueurs différentes.
ACTGG -> ACATGG : 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 $M = \text{« actgg »}$, 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 $n^k m^2$: le temps d'exécution 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 1^{er} dictionnaire.

Pour i allant de 2 à l

 Pour chaque modèle M de longueur $i-1$

 Pour chaque occurrence $occ = (n^{\circ} \text{ séquence}, \text{position})$ 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 $occ_{\text{nouv}} = (n^{\circ} \text{ seq}, \text{position})$ 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 I existe
Afficher dictionnaire I.
```

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
fini
    identifier x ;

    //extension sans erreur.
    Si Mx n'existe pas dans le dictionnaire i, alors ajouter Mx au dictionnaire i.
fini
    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 fini
    //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. fini
        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 : $occ = (N^{\circ}seq, pos)$

Occurrence approchée avec des erreurs de substitution. $Occ = (N^{\circ}seq, pos, s)$

Occurrence approchée avec des erreurs de substitution, d'insertion et de délétion. $Occ = (N^{\circ}seq, pos, s, i, d, last)$

Quand on est en train de créer un nouveau couple(modèle, occurrence) « 2^e 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

créer_le_dictionnaire_initial //toutes les occurrences du modèle vide.

Pour it allant de 1 à l :

Pour chaque modèle M :

Pour chaque occurrence $occ = (N^{\circ}seq, pos, s, i, d, last)$:

$j = pos + i - d + it$

 traiter_substitution(M, occ, k, Data[N^oSeq], j)

 match_possible ← vrai

 traiter_match(M, occ, k, Data[N^oSeq], match_possible, j)

 traiter_deletion(M, occ, k, Data[N^oSeq])

 traiter_insertion(M, occ, k, Data[N^oSeq], match_possible, j)

FinPour

FinPour

 Supprimer_modeles_ne_verifiant_pas_quorum_dans_dictionnaire_en_cours_de_construction(100 % par défaut)

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

 destruire_dictionnaire(i-1)

FinPour

PROCÉDURE traiter_substitution($\delta M, \delta occ = (N^{\circ}seq, pos, s, i, d, last), \delta k, \delta S, \delta j$)

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

Switch(last){

 subst :

Pour tout $x \neq S[j]$:

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

 Ajouter $occ_new = (N^{\circ}seq, pos, s+1, i, d, subst)$ à la liste des

occurrences de Mx.

FinPour

 insert :

Pour tout $x \neq S[j]$ et $x \neq S[j-1]$:

 Si le modele Mx n'existe pas dans le dictionnaire en cours de

construction Alors le créer Fin

 Ajouter $occ_nouv = (N^{\circ}Seq, pos, s+1, i, d, subst)$ aux occurrences

de Mx

FinPour

```

        match :
            Pour tout x != S[j] :
                Si le modele Mx n'existe pas dans le dictionnaire en cours de
construction Alors le créer Fin
                Ajouter occ_nouv = (N°seq, pos, s+1, i, d, subst) aux occurrences de
Mx
            FinPour

        del :
            rien à faire
    }

```

PROCÉDURE traiter_match(δM , $\delta \text{occ}=(N^\circ \text{seq}, \text{pos}, s, i, d, \text{last})$, δk , δS , $\phi \text{ match_possible}$, δj)

```

    Si l'occurrence ne peut pas être étendue Alors match_possible ← faux ; return Finsi
    Switch(last){
        subst :
            Si le modèle M.S[j] n'est pas présent dans le dictionnaire en cours de
construction Alors le créer Finsi
            Ajout occ_nouv = (N°seq, pos, s, i, d, match)

        insert :
            match_possible ← S[j-1] != S[j]
            Si (match_possible){
                Si le modèle M.S[j] n'est pas présent dans le dictionnaire en cours de
construction Alors le créer Finsi
                Ajout occ_nouv = (N°seq, pos, s, i, d, match)
            }

        match :
            Si le modèle M.S[j] n'est pas présent dans le dictionnaire en cours de
construction Alors le créer Finsi
            Ajout occ_nouv = (N°seq, pos, s, i, d, match)

        del :
            Si le modèle M.S[j] n'est pas présent dans le dictionnaire en cours de
construction Alors le créer Finsi
            Ajouter occ_nouv = (N°seq, pos, s, i, d, match)
    }

```

PROCÉDURE traiter_deletion(δM , $\delta \text{occ}=(N^\circ \text{seq}, \text{pos}, s, i, d, \text{last})$, δk , δS , δj)

```

    Si (s+i+d == k) Alors return Finsi
    Switch(last){
        subst :
            Pour tout x != S[j-1] :
                Si le modele Mx n'existe pas dans le dictionnaire en cours de
construction Alors le créer Fin
                Ajouter occ_nouv = (N°seq, pos, s, i, d+1, del) aux occurrences de
Mx
            FinPour

        insert :
            Rien à faire

        match :
            Pour tout x != S[j-1] :
                Si le modele Mx n'existe pas dans le dictionnaire en cours de
construction Alors le créer Fin

```

```

Mx
    Ajouter occ_nouv = (N°seq, pos, s, i, d+1, del) aux occurrences de
    FinPour
    del :
        Pour tout x pris dans l'alphabet ;
            Si le modele Mx n'existe pas dans le dictionnaire en cours de
construction Alors le créer Fin
            Ajouter occ_nouv = (N°seq, pos, s, i, d+1, del) aux occurrences de
Mx
    FinPour
}

PROCÉDURE traiter_insertion(δ M, δ occ=(N°seq, pos, s, i, d, last), δ k, δ S, δ j, δ
match_possible)
    Si (not match_possible) Alors return Finsi

    Si occ ne peut pas être étendu (immédiatement à droite) Alors nb_maximum_insertions ← k
- s - i - d FinSi
    Si M n'est pas présent dans le dictionnaire en cours de construction Alors le créer FinSi

    Pour e=1 à nb_maximum_insertions :
        Si le caractère correspondant à j+e n'existe pas Alors sortir boucle FinSi
        Ajouter occ_nouv = (N°seq, pos, s=0, i=e, d=0, last=insert)
    FinPour

PROCÉDURE creer_le_dictionnaire_initial(δ data, δ k)
    Créer le modèle vide et l'ajouter au dictionnaire courant. //unique entrée
    Pour k allant de 1 à n :
        Pour pos=1 à longueur(data[i]) : //ie séquence
            ajouter occ_nouv = (N°seq=i, pos, s=0, i=0, d=0, last=match) //match
modèle vide //occurrence vide
            Pour e=1 à k :
                Si le caractère correspondant à pos+e-1 n'existe pas Alors sortir
boucle FinSi
                Ajouter occ_nouv = (N°seq, pos, s, i+e, d, insert)
            FinPour
        FinPour
    FinPour

```

INFÉRENCE D'HAPLOTYPES

Génération de données d'haplotypes : onéreuse et longue.
 Palliatif : écrire un algo.

Exemple : Codage d'un génotype :

n loci.

Vecteur de taille n.

élément du vecteur : codage:nombre d'allèles mineurs :

si b est l'allèle mineur :

bb	2
bB, Bb	1
BB	0

Exemple de génotype :

2	1	1	0	1	2
---	---	---	---	---	---

paires d'haplotypes explicatifs (h1, h2) → g
 codage : b → 1
 B → 0

1	□	□	0	□	1
1	□	□	0	□	1

Si on a des positions ambiguës (généotype = 1) : 2^p paires explicatives pour ce génotype.

L'inférence d'haplotypes *in silico* repose sur le principe de parcimonie :
 objectif : maximiser le nb d'haplotypes partagés par les individus de la population concernée.

```

PROCÉDURE inférence_haplotypes_EM(db n:entier, db g:type_matrice, db seuil : réel, q
h:type_matrice, db nb_etapes_maximum:entier)
  initialiser les fréquences d'haplotypes
  calculer les probas des génotypes.
  convergence ← faux ; nbre ← 1
  vraisemblance_prec ← valeur_min

  TantQue (non convergence et nbre <= nb_etapes_maximum)
    incr(nbre)
    maximisation()
    vraisemblance ← estimation_espérance()
    convergence ← ( | vraisemblance - vraisemblance_prec | / vraisemblance_prec ) <=
seuil

    Si (non convergence) Alors vraisemblance_prec ← vraisemblance ; freq_prec ← freq ;
proba_prec ← proba ; FinSi
  Fin TantQue
  
```

Initialiser les fréquences d'haplotypes :

Haplotype : un identifiant, et un compteur.

Un vecteur g[i] pour chaque individu, contenant 0, 1 ou 2 dans chaque case. Une case correspondant à un SNP.

Dans l'ensemble de tous les haplotypes susceptibles d'entrer dans l'explication du génotype d'un individu de la population étudiée, attribuer p à chaque haplotype.

$$p = \frac{1}{nb_{haplotypes}}$$

| H | = nb_haplotypesusceptibles

Des indiv ont leurs génotypes expliqués par 1 paire d'haplotypes, d'autres par 12 paires... → on utilise des listes chaînées.

On aimerait accéder directement à la fréquence de h (où h est un haplotype en particulier).

// On alloue un tableau en dynamique, ou on dit que h correspond à une adresse vers une case où on a la fréquence.

Calcul de proba d'un génotype : calculé en fonction du nb d'haplotypes explicatifs.

→ on additionne les probas des explications.

Expl :

Génotype d'un indiv explicable par :

(h1, h2), (h7, h5), (h7, h15) et (h13, h13)

$g[i] = 2 \cdot \text{freq}(h1) \cdot \text{freq}(h2) + 2 \cdot \text{freq}(h7) \cdot \text{freq}(h5) + 2 \cdot \text{freq}(h7) \cdot \text{freq}(h15) + [\text{freq}(h13)]^2$ // on

multiplie par 2 car on si un indiv est expliqué par (h2,h1), il l'est aussi par (h1,h2)

calculProbasGénotypes :

pour chaque génotype g_{indiv} :


```

p ← 0
Pour chaque paire explicative (h1, h2) de gindiv :
    p1 ← fréquence de l'haplotype h1
    p2 ← fréquence de l'haplotype h2
    Si (h1 == h2)
        Alors ppart ← p12
        Sinon ppart ← 2.p1.p2
    Finsi
FinPour
FinPour

```

Expl :

h1,h2 → g1 //(h1,h2) explique g1
 nb individus qui ont le mm génotype : 10.

h1,h3 → g2 30 indiv
 h4,h4 → g2 50 indiv
 nbIndiv pour g2 : 80

h1,h1 → g3 30 indiv
 h5,h6 → g3 70 indiv
 nbIndiv pour g3 : 100

→ Total : 190 indiv

it : numéro d'itérations de l'algorithme EM.

freq^{it}(h1) = voir formule en dessous.

$$freq^{it}(h1) = 1/2 \cdot \left(\frac{10}{190} \cdot \frac{2 \cdot freq^{(it-1)}(h1) \cdot freq^{(it-1)}(h2)}{p^{(it-1)}(g1)} + \frac{30}{190} \cdot \frac{2 \cdot freq^{(it-1)}(h1) \cdot freq^{(it-1)}(h3)}{p^{(it-1)}(g2)} + \frac{30}{190} \cdot \frac{2 \cdot freq^{(it-1)}(h1)^2}{p^{(it-1)}(g3)} \right)$$

PROCÉDURE maximisation()

/*

préconditions :

Chaque haplotype est associé à la liste des génotypes dans l'explication desquels il est susceptible d'intervenir.

Le calcul qui va avoir lieu concerne l'étape it.

Pour l'étape it-1, les fréquences des haplotypes sont connues. (freq_prec).

Pour l'étape it-1, les probabilités des génotypes sont connues. (proba_prec).

N est le nombre total d'individus

Ngeno est le nombre d'individus possédant le génotype geno.

postcondition :

Pour l'étape it, les fréquences des haplotypes sont connus

*/

Pour chaque haplotype h1 :

récupérer freq_prec(h1)

freq ← 0

Pour chaque génotype geno susceptible d'être expliquée par h1 :

Si geno = (h1, h1)

Alors contribution ← 2.[freq_prec(h2)² / proba_prec(geno)] . [Ngeno / N]

Sinon //geno = (h1, h2)

Alors contribution ← 2.[freq_prec(h1) . freq_prec(h2) / proba_prec(geno)] . [

Ngeno / N]

FinSi

freq ← freq + contribution

```

    FinPour
    freq ← ½ freq
FinPour

```

```

PROCÉDURE estimation_esperance()

```

```

/*

```

```

Préconditions :

```

```

    Le calcul qui va avoir lieu concerne l'étape it
    Les fréquences des haplotypes sont connues pour l'étape it-1.
    Ngeno est le nombre d'individus possédant le génotype geno.

```

```

Postcondition :

```

```

    Les probas des génotypes sont connues à l'étape it.
    A vraisemblance des données a été calculée.

```

```

*/

```

```

LogLikelihood ← 0

```

```

Pour chaque génotype geno :

```

```

    p ← 0

```

```

    Pour chaque paire explicative (h1,h2) de geno :

```

```

        p1 ← freq(h1) ; p2 ← freq(h2)

```

```

        Si(h1 == h2)

```

```

            Alors ppart ← p12

```

```

        Sinon

```

```

            ppart ← 2.p1p2

```

```

        FinSi

```

```

        p ← p + ppart

```

```

    FinPour

```

```

    // La proba du génotype geno vient d'etre calculée

```

```

    loglikelihood ← loglikelihood + Ngeno.log(p)

```

```

FinPour

```

```

Rappels :

```

```

log(ab) = log(a) + log(b)

```

```

log(ab) = b.log(a)

```