Recherche de motifs : Algorithms randomisés

Cours 5

Plan du cours

- Les algorithmes randomisés
- Greedy Profile Motif Search
- Random Projections

Randomized Algorithms

- Les algorithmes randomisés prennent des décisions aléatoires plutôt que déterministes.
- L'algorithme s'exécute **différemment** à chaque fois.
- Ils sont couramment utilisés dans situations où aucun algorithme exact et rapide est connu.
- Nous allons voir deux algorithmes randomisés pour la recherche de motifs
  - Greedy Profile Motif Search
  - Random Projections

Greedy Profile Motif Search

1. Sélectionnez des positions de départ aléatoirement s = (s_1, …,s_t).
2. Créez un profil Pr à partir des séquences de ces positions de départ.
3. Trouver le P-most-probable k-mers **a** pour chaque séquence et changer la position de départ à la position de départ de **a**.
4. Calculer un nouveau profil basé sur le nouveaux positions de départs
5. procéder jusqu'à ce qu'on ne peut plus changer le Profil ou augmenter le score(s, DNA).

Exemple: Greedy Profile Motif Search

1. Sélectionnez des positions de départ aléatoirement $s = (s_1, \ldots, s_t)$.

$s = (6, 9, 1, 5, 4)$; $k = 8$

```
S1 TCATTCTCCGTTATGTAA

S2 TCTGAAGGGCCTCAGTAG

S3 GTTAAACTAGCCGTTTTC

S4 GATCGACATTGGCCCGCG

S5 CAAGGGCCTCTTCGATGG
```

Exemple: Greedy Profile Motif Search

2. Créez un profil Pr à partir des séquences de ces positions de départ.

S1 **CTCCGTTA**
S2 **GCCTCAGT**
S3 **GTTAAACT**
S4 **GACATTGG**
S5 **GGGCCTCT**

```
A [0. 1. 0. 2. 1. 2. 0. 1.]
C [1. 1. 3. 2. 2. 0. 2. 0.]
G [4. 1. 1. 0. 1. 0. 2. 1.]
T [0. 2. 1. 1. 1. 3. 1. 3.]
```

Pour éviter les valeurs à zéro nous pouvons utiliser un pseudocount

```
A [0+1. 2. 1. 3. 2. 3. 1. 2.]
C [1+1. 2. 4. 3. 3. 1. 3. 1.]
G [4+1. 2. 2. 1. 2. 1. 3. 2.]
T [0+1. 3. 2. 2. 2. 4. 2. 4.]
```

Par contre, nous allons laisser les valeurs à zéro pour éliminer rapidement les motifs peu probables

Exemple: Greedy Profile Motif Search

Transformer Pr dans une Matrice de Probabilité (PWM) en divisant chaque élément par la somme de colonnes

Pr =

A	4	7	3	0	1	0
C	1	0	4	5	3	0
T	1	1	0	0	2	7
G	2	0	1	3	2	1

PWM =

A	4/8	7/8	3/8	0/8	1/8	0/8
C	1/8	0/8	4/8	5/8	3/8	0/8
T	1/8	1/8	0/8	0/8	2/8	7/8
G	2/8	0/8	1/8	3/8	2/8	1/8

Exemple: Greedy Profile Motif Search

3. Trouver le P-most-probable k-mers **a** pour chaque séquence i et changer la position de départ s_i à la position de départ de **a**.

➔ Calculer la probabilité d'une séquence de taille k selon PWM

Prob(aaacct|PWM) = ???

➔ Calculer la probabilité d'une séquence de taille k selon la PWM

Prob(AAACCT|PWM) = 1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8 = .033646

A	**1/2**	**7/8**	**3/8**	0	1/8	0
C	1/8	0	1/2	**5/8**	**3/8**	0
T	1/8	1/8	0	0	1/4	**7/8**
G	1/4	0	1/8	3/8	1/4	1/8

Prob(ATACAG|PWM) = 1/2 x 1/8 x 3/8 x 5/8 x 1/8 x 1/8 = .001602

➔ P-most-probable k-mers

- Pour chaque séquence i trouver le “P-most-probable k-mer”, celui ayant la plus grande probabilité selon PMW

sequence = ctataaaccttacatc.

PWM =

A	1/2	7/8	3/8	0	1/8	0
C	1/8	0	1/2	5/8	3/8	0
T	1/8	1/8	0	0	1/4	7/8
G	1/4	0	1/8	3/8	1/4	1/8

c t a t a a a c c t t a c a t c

c **t a t a a a** c c t t a c a t c

c t **a t a a a c** c t t a c a t c

➔ P-most-probable k-mers

- Calculer la probabilité de tous les k-mers dans la séquence i

sequence = ctataaaccttacatc.

String, Highlighted in Red	Calculations	Prob(a|PWM)
ctataaaccttacat	1/8 x 1/8 x 3/8 x 0 x 1/8 x 0	0
ctataaaccttacat	1/2 x 7/8 x 0 x 0 x 1/8 x 0	0
ctataaaccttacat	1/2 x 1/8 x 3/8 x 0 x 1/8 x 0	0
ctataaaccttacat	1/8 x 7/8 x 3/8 x 0 x 3/8 x 0	0
ctataaaccttacat	1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8	.0336
ctataaaccttacat	1/2 x 7/8 x 1/2 x 5/8 x 1/4 x 7/8	.0299
ctataaaccttacat	1/2 x 0 x 1/2 x 0 1/4 x 0	0
ctataaaccttacat	1/8 x 0 x 0 x 0 x 0 x 1/8 x 0	0
ctataaaccttacat	1/8 x 1/8 x 0 x 0 x 3/8 x 0	0
ctataaaccttacat	1/8 x 1/8 x 3/8 x 5/8 x 1/8 x 7/8	.0004

➔ P-most-probable k-mers

- Le k-mer le plus probable est **aaacct** avec probabilité 0.0336, et commence à la position 5

String, Highlighted in Red	Calculations	Prob(a|PWM)
ctataaaccttacat	1/8 x 1/8 x 3/8 x 0 x 1/8 x 0	0
ctataaaccttacat	1/2 x 7/8 x 0 x 0 x 1/8 x 0	0
ctataaaccttacat	1/2 x 1/8 x 3/8 x 0 x 1/8 x 0	0
ctataaaccttacat	1/8 x 7/8 x 3/8 x 0 x 3/8 x 0	0
ctataaaccttacat	**1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8**	**.0336**
ctataaaccttacat	1/2 x 7/8 x 1/2 x 5/8 x 1/4 x 7/8	.0299
ctataaaccttacat	1/2 x 0 x 1/2 x 0 1/4 x 0	0
ctataaaccttacat	1/8 x 0 x 0 x 0 x 0 x 1/8 x 0	0
ctataaaccttacat	1/8 x 1/8 x 0 x 0 x 3/8 x 0	0
ctataaaccttacat	1/8 x 1/8 x 3/8 x 5/8 x 1/8 x 7/8	.0004

➔ P-most-probable k-mers

- On doit calculer le kmer le plus probable pour toutes les séquence et recalculer le vecteur s

ctataaacgttacatc

atagcgattcgactg

cagcccagaaccct

cggtgaaccttacatc

tgcattcaatagctta

tgtcctgtccactcac

ctccaaatcctttaca

ggtctacctttatcct

S=(5, 1, 9, 5, 9, 1, 7, 5)

4. Calculer un nouveau profil basé sur le nouveaux positions de départs

S=(5, 1, 9, 5, 9, 1, 7, 5)

ctataaacgttacatc

atagcgattcgactg

cagcccagaaccct

cggtgaaccttacatc

tgcattcaatagctta

tgtcctgtccactcac

ctccaaatcctttaca

ggtctacctttatcct

1	a	a	a	c	g	t
2	a	t	a	g	c	g
3	a	a	c	c	c	t
4	g	a	a	c	c	t
5	a	t	a	g	c	t
6	g	a	c	c	t	g
7	a	t	c	c	t	t
8	t	a	c	c	t	t
A	5/8	5/8	4/8	0	0	0
C	0	0	4/8	6/8	4/8	0
T	1/8	3/8	0	0	3/8	6/8
G	2/8	0	0	2/8	1/8	2/8

5. procéder jusqu'à ce que nous ne puissions plus changer le Profil.

Nouveau Profile

A	5/8	5/8	4/8	0	0	0
C	0	0	4/8	6/8	4/8	0
T	1/8	3/8	0	0	3/8	6/8
G	2/8	0	0	2/8	1/8	2/8

Profile precedent

A	1/2	7/8	3/8	0	1/8	0
C	1/8	0	1/2	5/8	3/8	0
T	1/8	1/8	0	0	1/4	7/8
G	1/4	0	1/8	3/8	1/4	1/8

Rouge = les fréquences ont augmenté
Bleu = les fréquences ont diminué

GreedyProfileMotifSearch(DNA, t, n, k)

Randomly select starting positions $s = (s_1, \ldots, s_t)$ from DNA

bestScore = 0

while Score(s, DNA) > bestScore

form profile Pr from s

bestScore = Score(s, DNA)

for i = 1 to t

find a P-most probable k-mer **a** from the ith sequence

s_i = starting position of **a**

return bestScore, s

Le score est la somme de fréquences/probabilités maximal de chaque position

Analise : Greedy Profile Motif Search

- Puisque nous choisissons des positions de départ au hasard, il y a peu de chances que notre choix soit proche d'un motif optimal
- Il est peu probable que les positions de départ aléatoires nous mènent à la bonne solution.
- En pratique, cet algorithme est **exécuté plusieurs fois** dans l'espoir que les positions de départ aléatoires soient proches de la solution optimale simplement par hasard.
- Nous pouvons aussi abandonner les positions de départ qui produisent des motifs peu complexes.

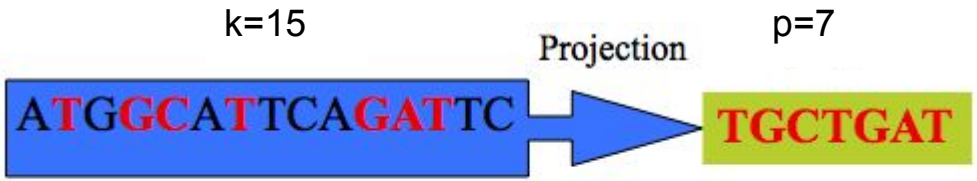
```
GreedyProfileMotifSearchIterative(DNA, t, n, k , It)
    Scores = []
    Positions = [[]]

    For i = 1 to It

        score, s = GreedyProfileMotifSearch(DNA, t, n, k)
        Scores[i] = score
        Positions[i] = s
    J = maxIndex(Scores)
    return Position[J]
```

Random Projection

- Certaines instances d'un motif s'accordent sur un sous-ensemble de positions car il peut exister de **variation/mutation**.
- Cependant, on ne sait pas comment trouver ces positions «non mutées».
- Nous pouvons sélectionner au hasard un sous-ensemble de positions dans le motif créant une projection du motif.
- Recherchez cette projection dans l'espoir que les positions sélectionnées ne soient pas affectées par des mutations.

Projection

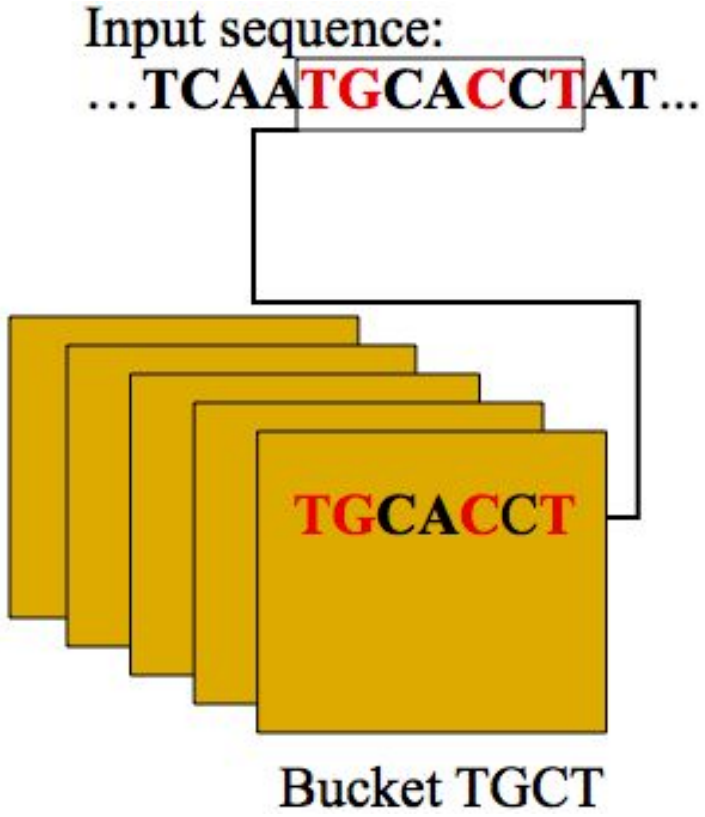
- Choisissez k positions dans une sequence de longueur L.
- Concaténer les K nucléotides choisi pour former une clé de taille k.
- Cela peut être considéré comme une projection de l'espace L-dimensionnel sur k-dimensionnel



Projection = (2, 4, 5, 7, 11, 12, 13)

Random Projection algorithm

- Sélectionnez p parmi k positions uniformément
- Pour chaque k-tuple dans les séquences d'entrée trouver son bucket p
- Récupérez le motif du bucket enrichi qui contient de nombreux k-tuples.



Random Projection algorithm

```
S1 TCATTCTCCGTTATGTAA

S2 TCTGAAGGGCCTCAGTAG

S3 GTTAAACTAGCCGTTTTC

S4 GATCGACATTGGCCCGCG

S5 CAAGGGCCTCTTCGATGG
```

p=4 ; k= 8

Sélectionnez p parmi k positions uniformément

Projection = (2, 4, 5, 7)

Random Projection algorithm

p=4 ; k= 8 Projection = (2, 4, 5, 7)

S1 **TCATTCTC**CGTTATGTAA

S2 TCTGAAGGGCCTCAGTAG

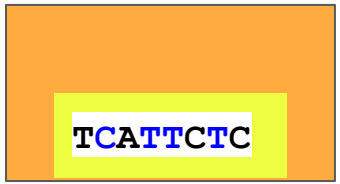
S3 GTTAAACTAGCCGTTTTC

S4 GATCGACATTGGCCCGCG

S5 CAAGGGCCTCTTCGATGG

Pour chaque motif k dans les séquences d'entrée

trouver son bucket p



CTTT

Random Projection algorithm

S1 TCATTCTCCGTTATGTAA

S2 TCTGAAGGGCCTCAGTAG

S3 GTTAAACTAGCCGTTTTC

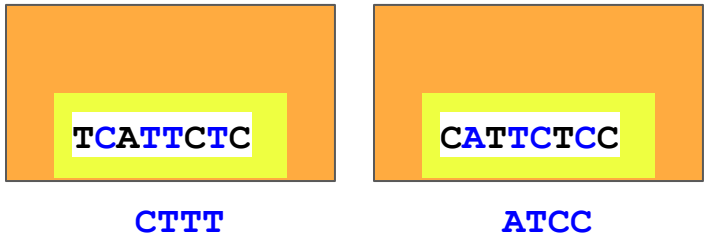
S4 GATCGACATTGGCCCGCG

S5 CAAGGGCCTCTTCGATGG

p=4 ; k= 8 Projection = (2, 4, 5, 7)

Pour chaque motif k dans les séquences d'entrée

trouver son bucket p



Random Projection algorithm

S1 TCATTCTCCGTTATGTAA

S2 TCTGAAGGG**CCTTTCTA**G

S3 GTTAAACTAGCCGTTTTC

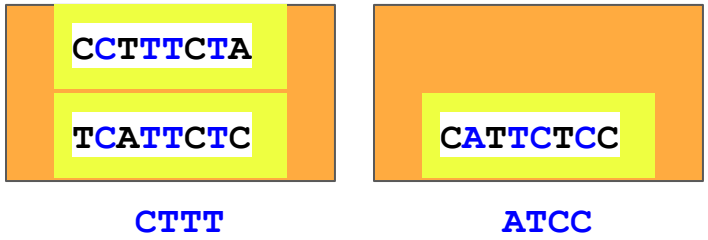
S4 GATCGACATTGGCCCGCG

S5 CAAGGGCCTCTTCGATGG

p=4 ; k= 8 Projection = (2, 4, 5, 7)

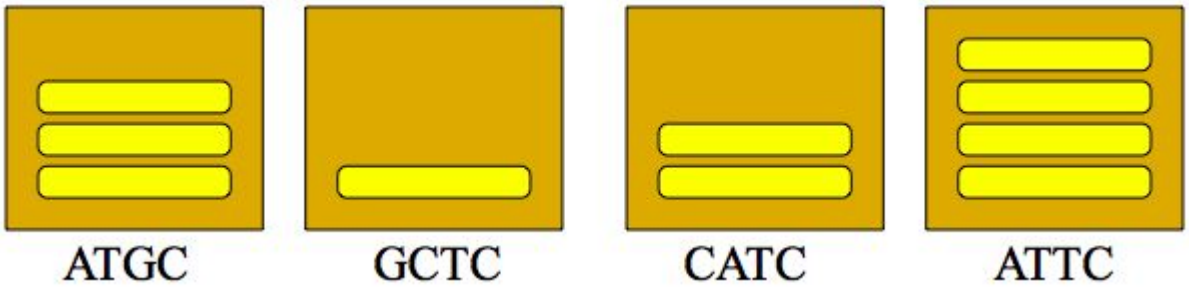
Pour chaque motif k dans les séquences d'entrée

trouver son bucket p



Random Projection algorithm

- Certaines projections ne parviendront pas à détecter les motifs, mais si nous essayons plusieurs d'entre elles, la probabilité qu'un des bucket se remplisse augmente.



Random Projection algorithm

```
randomProjection(DNA, t, n, k, p)
    Bucket = {}
    p = createInitialProjection(p, k)
    Foreach motif of size k in DNA
        Key = generateProjection(motif, p)
        Bucket [key] = motif
    return Bucket
```

Random Projection algorithm

- Une seule itération va choisir une projection de taille p->k aléatoire.
- Nous pouvons relancer l'algorithme plusieur fois
- Nous évaluons donc les N meilleurs Buckets, à travers ses scores, et nous sélectionnons le motif avec le plus grand score
- Attention : Éliminer les motifs peu complexe