M2 - STL

Algorithmes sur les séquences en bioinformatique

Cours 4: Algorithmes exactes de recherche de motifs, CONSENSUS et PatternBranching

Alessandra Carbone
Université Pierre et Marie Curie

Plan

- Motifs dans un texte aléatoire
- Régulation des gènes
- Motifs régulateurs
- le problème du Gold Bug
- Le problème Motif Finding
- Brute Force Motif Finding
- Le problème Median String Search
- Arbres de recherche
- Branch-and-Bound Motif Search
- Branch-and-Bound Median String Search
- Consensus et Pattern Branching: algorithme gourmand
- PMS: Exhaustive Motif Search

Séquences aléatoires

atgaccgggatactgataccgtatttggcctaggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaactttccgaatactgggcataaggtaca
tgagtatccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgcccagggtccga
gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag
gtcaatcatgttcttgtgaatggatttttaactgagggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtactgatggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca

Insertion de motifs : **AAAAAAAGGGGGGG**

atgaccgggatactgatAAAAAAAAGGGGGGGgcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaactttccgaataAAAAAAAAGGGGGGGa
tgagtatccctgggatgacttAAAAAAAAGGGGGGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgAAAAAAAAGGGGGGGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAAAAAAAAGGGGGGGcttatag
gtcaatcatgttcttgtgaatggatttAAAAAAAAGGGGGGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtAAAAAAAAGGGGGGGcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttAAAAAAAAGGGGGGGctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatAAAAAAAAGGGGGGGaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttAAAAAAAAGGGGGGGa

Ou sont les motifs insérés ?

atgaccgggatactgatAAAAAAAAGGGGGGGGcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaactttccgaatatAAAAAAAAGGGGGGGa

tgagtatccctgggatgacttAAAAAAAAGGGGGGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgAAAAAAAAGGGGGGGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAAAAAAAAGGGGGGGcttatag

gtcaatcatgttcttgtgaatggatttAAAAAAAAGGGGGGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAAAAAAAAGGGGGGGcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAAAAGGGGGGGctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatAAAAAAAAGGGGGGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttAAAAAAAAGGGGGGGa

Insertion de motifs **AAAAAAGGGGGGG** avec 4 substitutions

atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaactttccgaataCAATAAAACGGCGGGa

tgagtatccctgggatgacttAAAATAATGGAGTGGTgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgCAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatATAATAAAGGaaGGGcttatag

gtcaatcatgttcttgtgaatggatttAACAATAAGGGCTGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtATAAACAAGGaGGGCcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAATAGGGaGCCctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatACTAAAAAGGaGCGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttACTAAAAAGGaGCGGa

Où sont les motifs ?

atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaactttccgaatacAATAAAACGGCGGGa

tgagtatccctgggatgacttAAAATAATGGaGtGGTgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgCAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAATAAAGGaaGGGcttatag

gtcaatcatgttcttgtgaatggatttAACAATAAGGGCtGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAtAAACaAGGaGGGCcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAATAGGGaGCCctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGCGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGCGGa

Pourquoi chercher les motifs (15,4) est compliquée ?

atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaactttccgaatacAATAAAACGGCGGGa

tgagtatccctgggatgacttAAAATAATGGaGtGGTgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgCAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAATAAAGGaaGGGcttatag

gtcaatcatgttcttgtgaatggatttAACAATAAGGGCtGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAtAAACaAGGaGGGCcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAATAGGGaGCCctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGCGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGCGGa

AgAAgAAAGGttGGG

..|..|||.|..|||

cAATAAAACGGcGGG

Problème

Trouver un motif dans un texte aléatoire

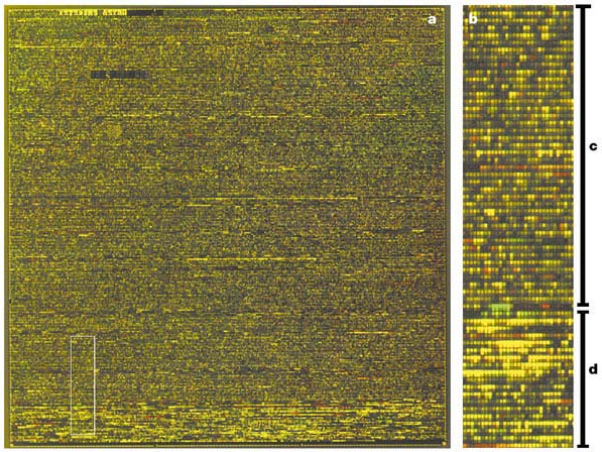
- 20 séquences “aléatoires” (e.g. 600 nt de longueur)
- chaque séquence contient un motif inséré de longueur 15,
- chaque motif apparaît avec 4 mismatches, et on l’appelle motif (15,4).

Combinatoire de la régulation des gènes

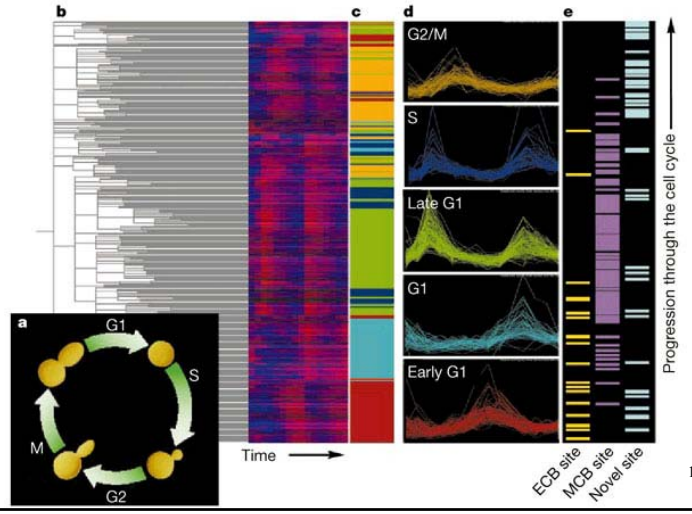
Les expériences utilisant les micropuces d’ADN ont montré que quand un gène X est supprimé (“knocked out”), 20 autres gènes ne sont pas exprimés

Comment c’est possible qu’un seul gène puisse avoir un effet tellement drastique ?

Nouvelles technologies : les puces à ADN



Exemple: analyse du cycle cellulaire de la levure



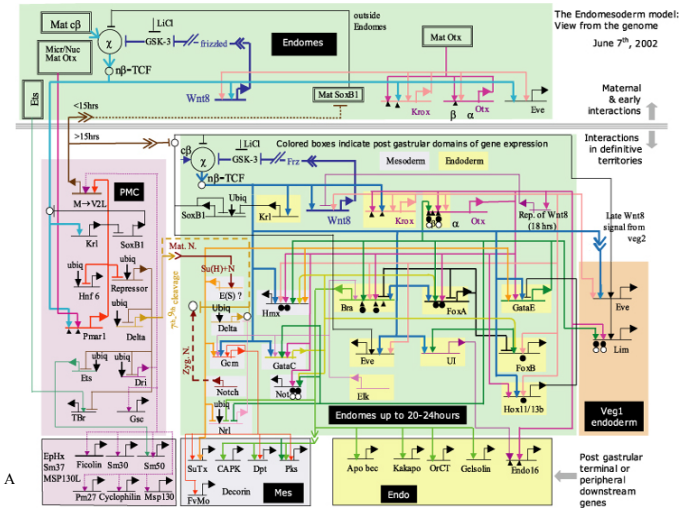
Protéines régulatrices

- Gène X code pour une protéine régulatrice, appelée ***facteur de transcription*** (TF)
- Les 20 gènes non exprimés demandent la présence du TF codé par le gène X pour avoir une induction de leur transcription.
- Un seul TF peut réguler plusieurs gènes.

Identification de motifs

Trouver des motifs dans plusieurs régions régulatrices suggère une relation de régulation entre ces gènes.

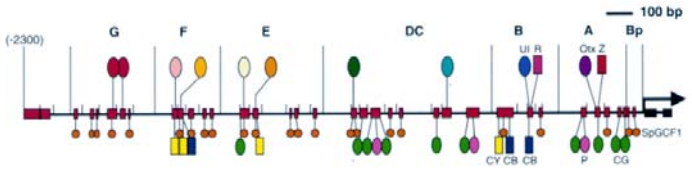
Un exemple de circuit de régulation : les gènes de développement de l'oursin



Régions régulatrices

- Chaque gène contient une région régulatrice (RR) typiquement située 100-1000 bp en amont du site de départ de la transcription.
- Localisés dans la RR il y a les ***sites de liaison des facteurs de transcription*** (TFBS), nommé ***motifs***, spécifiques d'un facteur de transcription donné.
- Les TFs influencent l'expression du gène en se liant a une location spécifique dans la région régulatrice respective.

Exemple : développement de l'oursin de mer

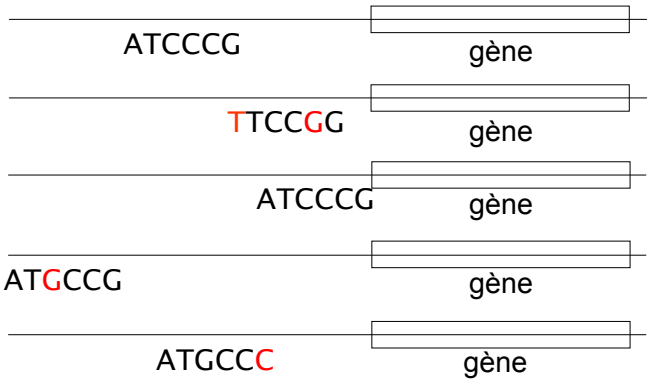


Région promotrice du gène endo16

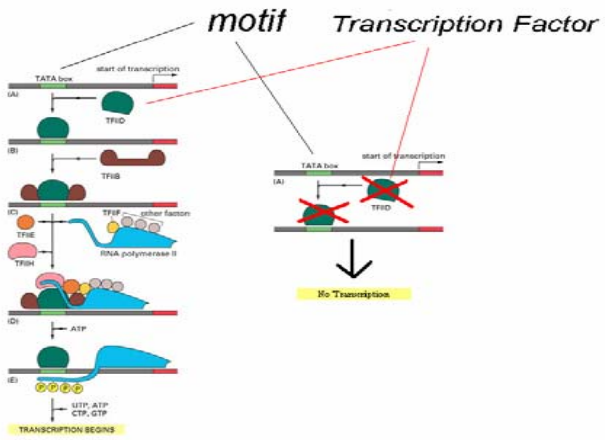
Sites de liaison des facteurs de transcription

- Un TFBS peut être localisé n'importe où dans la région régulatrice (RR)
- Un TFBS peut varier dépendamment des régions régulatrices parce que des bases non-essentielles peuvent avoir mutées.

Motifs et site de départ de la transcrition



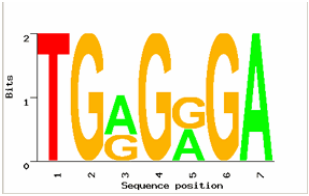
Facteurs de transcription et motifs



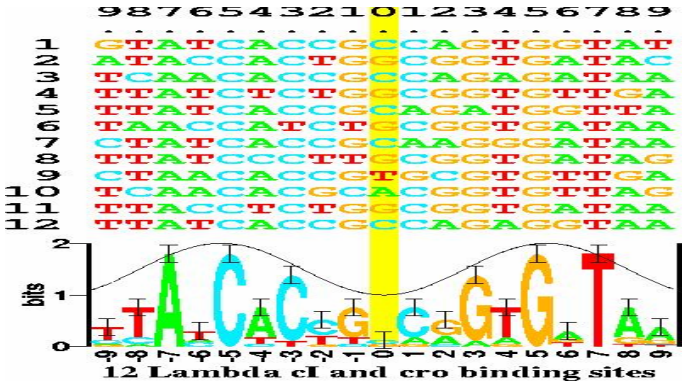
Logo du motif

- Les motifs peuvent muter sur des positions qui se révèlent n'être pas importantes
- Les 5 motifs dans 5 gènes différents ont muté a la position 3 et 5
- La représentation est appelée *logo du motif* et représente les régions conservées et les régions variables du motif.

TGGGGGA
TGAGAGA
TGGGGGA
TGAGAGA
TGAGGGA



Logo de motif: un exemple



(http://www-lmmb.ncifcrf.gov/~toms/sequencelogo.html)

Identification de motifs: complications

- On ne connaît pas la séquence du motif
- On ne connaît pas où il se trouve le motif par rapport au début du gène
- Motifs peuvent différer de gène à gène
- Comment les distinguer des motifs "aléatoires" ?

Analogie de la recherche de motifs

Le problème de la recherche de motifs est **similaire** au problème posé par Edgar Allan Poe (1809 – 1849) dans son compte *Gold Bug*

Le problème du Gold Bug

- Etant donné un message secret :

```
53++!305))6*;4826)4+.)4+);806*;48!8`60))85;]8*:+*8!83(88)5*!;
46(;88*96*?;8)*+(;485);5*!2:*+(;4956*2(5*-4)8`8*; 4069285);)6
!8)4++;1(+9;48081;8:8+1;48!85;4)485!528806*81(+9;48;(88;4(+?3
4;48)4+;161;:188;+?;
```

- On veut déchiffrer le message encrypté dans le fragment

Suggestion pour le problème de Gold Bug

- Le message encrypté est écrit en anglais
- Chaque symbole correspond a une lettre de l'alphabet anglais
- La ponctuation n'est pas encodé

Le problème du Gold Bug : comptage de symboles

- Approche naïve a la solution du problème:
  - Compter la fréquence de chaque symbole dans le message encrypté.
  - Trouver la fréquence de chaque lettre dans l'alphabet anglais.
  - Comparer les fréquences obtenues, chercher une corrélation et trouver une correspondance entre symboles et lettres de l'alphabet.

Fréquences des symboles dans le message Gold Bug

- **Message du Gold Bug** :

Symbole	8	;	4	)	+	*	5	6	(	!	1	0	2	9	3	:	?	`	-	]	.
Fréquence	34	25	19	16	15	14	12	11	9	8	7	6	5	5	4	4	3	2	1	1	1

- **Langue anglaise**:

e t a o i n s r h l d c u m f p g w y b v k x j q z

plus fréquent  moins fréquent

La décodification du message du Gold Bug : premier tentative

- La correspondance entre le symbole plus fréquent et la lettre plus fréquente de l'alphabet:

```
sfiilfcsoorntaeuroaikoaiotecrntaeleyrcooestvenpinelefheeosnlt
arhteenmrnwteonihtaesotsnlupnihtamsrnuhsnbaoeyentacrmuesotorl
eoaiitdhimtaecedtepeidtaelestaoaeslsueecrnedhimtaetheetahiwfa
taeoaitdrdtpdeetiwt
```

- Le résultat n'a pas de sens

Le problème du problème du Gold Bug : comptage des *l*-tuples

- Une meilleure approche :
  - Examiner les fréquences des *l*-tuples, combinaisons de 2 symboles, 3 symboles, etc.
  - "The" est la 3-tuple plus fréquente en anglais et ";48" est la 3-tuple plus fréquente dans le texte encrypté.
  - Inférer les symboles non connus en examinant *l*-tuples fréquents

Le problème du Gold Bug : le clue ;48

La correspondance entre "the" et ";48" et la substitution de toutes les occurrence des symboles:

```
53++!305))6*the26)h+.)h+)te06*the!e`60))e5t]e*:+*e!e3(ee)5*!t
h6(tee*96*?te)*+(the5)t5*!2:*+(th956*2(5*h)e`e*th0692e5)t)6!e
)h++t1(+9the0e1te:e+1the!e5th)he5!52ee06*e1(+9thet(eeth(+?3ht
he)h+t161t:1eet+?t
```


Décodification du message du Gold Bug : deuxième tentative

- Inférer :

```
53++!305))6*the26)h+.)h+)te06*the!e`60))e5t]e*:+*e!e3(ee)5*!t
h6(tee*96*?te)*+(the5)t5*!2:*+(th956*2(5*h)e`e*th0692e5)t)6!e
)h++t1(+9the0e1te:e+1the!e5th)he5!52ee06*e1(+9thet(eeth(+?3ht
he)h+t161t:1eet+?t
```

- "thet(ee" devient avec forte probabilité "the tree"
  - Inférer : "(" = "r"
- "th(+?3h" devient "thr+?3h"
  - Est-ce qu'on peux deviner "+" et "?"?

Le problème de Gold Bug : solution

Après avoir reconstruit toutes les correspondances, le message finale est :

AGOODGLASSINTHEBISHOPSHOSTELINTHEDEVILSSEATWENYONEDEGRE
ESANDTHIRTEENMINUTESNORTHEASTANDBYNORTHMAINBRANCHSEVENT
HLIMBEASTSIDESHOOTFROMTHELEFTEYEOFTHEDEATHSHEADABEELINE
FROMTHETREETHROUGHTHESHOTFIFTYFEETOUT

La solution (continuation)

La ponctuation est importante:

A GOOD GLASS IN THE BISHOP'S HOSTEL IN THE DEVIL'S SEA,
TWENY ONE DEGREES AND THIRTEEN MINUTES NORTHEAST AND BY NORTH,
MAIN BRANCH SEVENTH LIMB, EAST SIDE, SHOOT FROM THE LEFT EYE OF
THE DEATH'S HEAD A BEE LINE FROM THE TREE THROUGH THE SHOT,
FIFTY FEET OUT.

Résoudre le problème du Gold Bug

- Conditions préalables pour trouver la solution :
  - On doit connaître les fréquences relatives des lettres seules, et des combinaisons de 2 ou 3 lettres en anglais.
  - La connaissance de tous les mots du dictionnaire anglais est souhaitée pour une inférence précise.

Recherche de motifs et le problème du Gold Bug : similarités

- Les nucléotides dans les motifs codent pour un message dans le langage "génétique". Les lettres dans le "the gold bug" codent pour un message en anglais.
- Pour résoudre le problème, nous analysons les fréquences des motifs dans le message de l'ADN/Gold Bug.
- La connaissance de motifs régulateurs connus rende le problème plus simple. La connaissance des mots dans le dictionnaire anglais aide à la résolution du problème du Gold Bug.

La recherche des motifs est un problème plus difficile que le **problème du Gold Bug**:

- Nous n'avons pas un dictionnaire complet de motifs.
- Le langage "génétique" n'a pas de grammaire standard.
- Seulement une petite fraction des séquences nucléotidiques encodent des motifs; la taille des données est énorme.

Le problème de la recherche des motifs

Etant donnée des séquences aléatoires d'ADN:

```
cctgatagacgctatctggctatccacgtacgtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtacgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtacgtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtacgtc
```

Trouver le motif qui est présent dans toutes ces séquences.

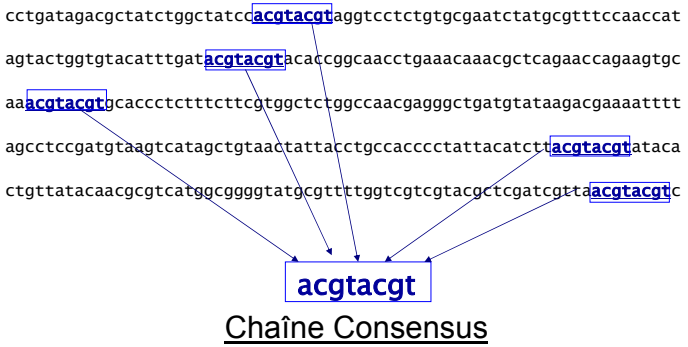
Le problème de la recherche des motifs

Information additionnelle:

- La séquence cachée est de longueur 8
- Le motif n'est pas le même dans chaque séquence à cause des substitutions

Le problème de la recherche des motifs

- Un motif sans mutations :



Le problème de la recherche des motifs

- Un motif avec 2 mutations:

cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat

agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc

aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt

agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca

ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc

Le problème de la recherche des motifs

Un motif a 2 mutations:

cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat

agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc

aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt

agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca

ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc

Est-ce que l'on peut toujours trouver un motif, avec 2 mutation ?

Définition des motifs

- Pour définir un motif, supposons de connaître où le motif commence dans la séquence
- Les positions du départ du motif dans les séquences peuvent être représentées par $\boldsymbol{s} = (s_1, s_2, s_3, \ldots, s_t)$

Motifs: Profiles et Consensus

Alignement

```
a G g t a c T t
C c A t a c g t
a c g t T A g t
a c g t C c A t
C c g t a c g G
```

Profile

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

Consensus

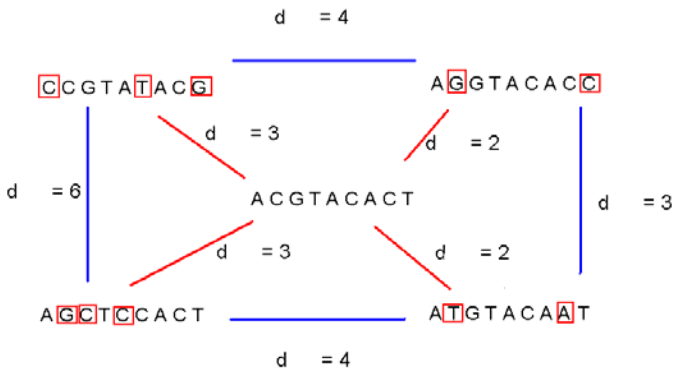
```
A C G T A C G T
```

- Aligner les motifs par leur index de départ
  $\boldsymbol{s} = (s_1, s_2, \ldots, s_t)$
- Construire la matrice profile avec les fréquences de chaque nucléotide dans les colonnes
- Le nucléotide consensus dans chaque position a le score plus élevé dans la colonne

Consensus

- Nous pensons au consensus comme a un motif “ancêtre”, qui est l’origine des motifs émergeants.
- La *distance* entre un motif réel et une séquence consensus est généralement plus petite que entre 2 motifs réels.

Consensus (continuation)



Evaluation des motifs

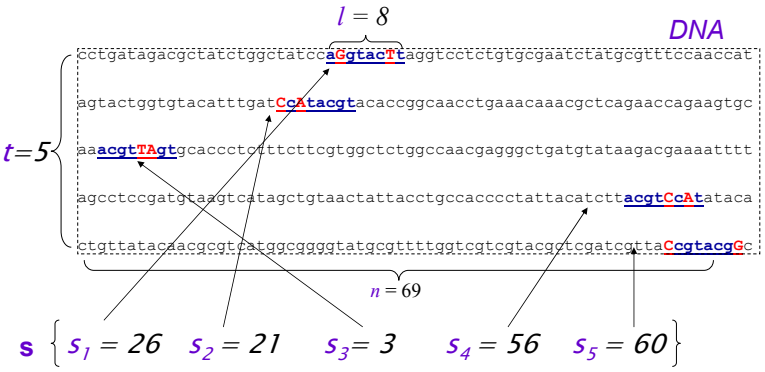
- Nous avons une idée de la séquence consensus, mais quel tôt de confiance on peut avoir dans elle ?
- Il faut introduire une **fonction de score** pour comparer les différentes propositions de séquence consensus et pour choisir la “meilleure”.

Terminologie

- ***t*** - nombre de séquences ADN
- ***n*** - longueur de chaque séquence ADN
- ***DNA*** – séquences ADN (tableau *t* x *n*)

- ***l*** - longueur du motif (***l***-mer)
- s_i - position du départ d’un ***l***-mer dans la séquence *i*
- **s**=(s_1, s_2,… s_t) - tableau des positions de départ du motif

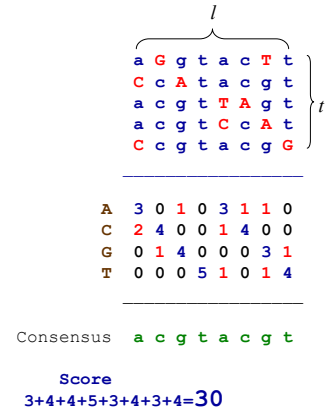
Paramètres



Score des motifs

- Etant donné **s** = (s_1, … s_t) et ***DNA***:

$$Score(\mathbf{s},DNA) = \sum_{i=1}^{l} \max_{k\in\{A,T,C,G\}} count(k,i)$$



Le problème de la recherche des motifs

- Si les positions de départ sont données **s**=(s_1, s_2,… s_t), trouver un consensus est facile même s'il y a des mutations parce qu'on peut simplement construire un profile (pour trouver le motif).
- mais… les positions de départ **s** ne sont pas connues d'habitude. Comment peut on trouver alors la matrice de profile la meilleure?

Le problème de la recherche des motifs: formulation

- <u>But</u>: étant donné un ensemble de séquences d'ADN, trouver un ensemble de ***l***-mers, un pour chaque séquence, que maximise le score de consensus.
- <u>Input</u>: une matrice ***t*** x ***n*** de sequences d'ADN (***DNA***), et ***l***, la longueur du motif a chercher
- <u>Output</u>: un tableau de ***t*** positions de départ **s** = (s_1, s_2, … s_t) qui maximisent *Score*(**s**,***DNA)***

Le problème de la recherche des motifs: algorithme brute force

- Calcule les scores pour toutes combinaisons possibles des positions de départ **s**
- Le meilleur score déterminera le meilleur profile et le motif consensus dans ***DNA***
- Le but est de maximiser le *Score*(**s**,***DNA***) en variant les positions de départ s_i, ou:

$$s_i = [1, \ldots, n-l+1]$$
$$i = [1, \ldots, t]$$

BruteForceMotifSearch

1. BruteForceMotifSearch(***DNA, t, n, l***)
2. ***bestScore*** ← 0
3. **for** each **s**=(s_1,s_2 , . . ., s_t) from (1,1 . . . 1) to ($n-l+1$, . . ., $n-l+1$)
4. **if** (*Score*(**s**,***DNA***) > ***bestScore***)
5. ***bestScore*** ← *score*(**s**, ***DNA***)
6. **bestMotif** ← (s_1,s_2 , . . . , s_t)
7. **return bestMotif**

Complexité de BruteForceMotifSearch

- On regarde $(n - l + 1)^t$ ensembles de positions de départ
- Pour chaque ensemble de positions de départ, la fonction de score est calculée avec ***l*** opérations, et la complexité devient
  $$l\,(n - l + 1)^t = O(l\,n^t)$$
- Cela signifie que pour ***t*** = 8, ***n*** = 1000, ***l*** = 10 on doit réaliser environs 10^{20} opérations – en qqs millions d'années.

Le problème de la recherche de la chaîne médiane (Median String Search Problem)

- Etant donné un ensemble de ***t*** séquences d'ADN trouver un motif qui apparaît dans toutes les ***t*** séquences avec un nombre minimale de mutations

Distance de Hamming

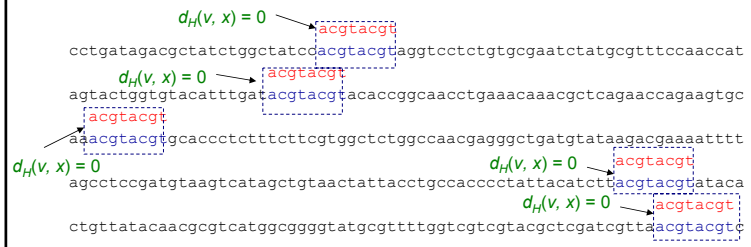
Distance de Hamming:

– $d_H(\boldsymbol{v},\boldsymbol{w})$ est le nombre de paires nucléotidiques que ne sont pas appariées dans l'alignement de ***v*** et ***w***. Par exemple:

$$d_H(\text{AAAAAA},\text{ACAAAC}) = 2$$

Distance totale : un exemple

- Etant donné ***v*** = "acgtacgt" et **s**



v est la sequence en rouge, *x* est la sequence en bleue

- $TotalDistance(\boldsymbol{v},\boldsymbol{DNA}) = 0$

Distance totale : un exemple

- Etant donné ***v*** = "acgtacgt" et **s**

v est la sequence en rouge, *x* est la sequence en bleue

- $TotalDistance(\boldsymbol{v},\boldsymbol{DNA}) = 1+0+2+0+1 = 4$

Distance totale : un exemple

– Pour chaque séquence d'ADN ***i***, calculer toutes $d_H(\boldsymbol{v}, \boldsymbol{x})$, où ***x*** est un ***l***-mer ayant la position de départ s_i ($1 \leq s_i \leq \boldsymbol{n} - \boldsymbol{l} + 1$)

– Trouver le $d_H(\boldsymbol{v}, \boldsymbol{x})$ minimum entre tous les ***l***-mers dans la séquence ***i***

– $TotalDistance(\boldsymbol{v},\boldsymbol{DNA})$ est la somme des distances de Hamming minimales des séquences d'ADN ***i***

$$TotalDistance(\boldsymbol{v},\boldsymbol{DNA}) = min_{\boldsymbol{s}}\, d_H(\boldsymbol{v}, \boldsymbol{s})$$

où **s** est l'ensemble des positions de départ $s_1, s_2, \ldots s_t$

Le problème de la chaîne médiane (Median String Problem) : formulation

- But: étant donné un ensemble de séquences d'ADN, trouver une chaîne médiane
- Input: une matrice de séquences d'ADN ***t*** x ***n***, et ***l***, la longueur d'un motif a chercher
- Output: une chaîne ***v*** de ***l*** nucléotides que minimises *TotalDistance(**v**,**DNA**)* sur toutes les chaînes de telle longueur

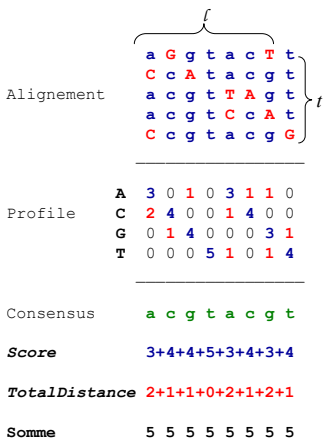
Median String Search Algorithm

1. MedianStringSearch (***DNA***, ***t***, ***n***, ***l***)
2. ***bestWord*** ← AAA…A
3. ***bestDistance*** ← ∞
4. **for** each *l*-mer **s from** AAA…A to TTT…T
   if *TotalDistance*(**s**,*DNA*) < ***bestDistance***
5. ***bestDistance*** ← *TotalDistance*(**s**,***DNA***)
6. ***bestWord*** ← **s**
7. **return** ***bestWord***

Motif Finding Problem ≡ Median String Search Problem

- Le *Motif Finding* est un problème de maximisation et le *Median String Search* est un problème de minimisation
- le *Motif Finding* et le *Median String Search* sont des problèmes computationellement équivalents
- Il faut montrer que minimiser la *TotalDistance* est équivalent a maximiser le *Score*

On regarde la même chose



- A chaque colonne ***i*** $Score_i + TotalDistance_i = t$
- Comme il y a ***l*** colonnes $Score + TotalDistance = l * t$

  et donc :

  $Score = l * t - TotalDistance$
- Comme $l * t$ est constant, la minimisation sur la droite est équivalente a la maximisation sur la gauche

Motif Finding Problem vs. Median String Search Problem

- Pourquoi reformuler le problème Motif Finding dans le problème Median String Search?
  - Le problème Motif Finding demande l'analyse de toutes les combinaisons de **s**. Cad $(n - l + 1)^t$ combinaisons!!!
  - Le problème Median String Search demande l'analyse de 4^l combinaisons pour ***v***. Ce nombre est relativement plus petit.

Motif Finding: amélioration du temps de calcul

On rappelle le BruteForceMotifSearch:

```
1.  BruteForceMotifSearch(DNA, t, n, l)
2.  bestScore ← 0
3.  for each s=(s1,s2 , . . ., st) from (1,1 . . . 1) to (n–l+1, . . ., n–l+1)
4.      if (Score(s,DNA) > bestScore)
5.          bestScore ← Score(s, DNA)
6.          bestMotif ← (s1,s2 , . . . , st)
7.  return bestMotif
```


Structuration de la recherche des motifs

- Comment peut-on implémenter la ligne?

  for each **s=**$(s_1,s_2 , . . ., s_t)$ from (1,1 . . . 1) to (***n-l***+1, . . ., ***n-l***+1)

- Nous avons besoin d'une méthode pour structurer et "bouger" efficacement dans l'espace des motifs possibles.
- Ce problème n'est pas trop différent que l'exploration de tous les nombres ayant *t*-digits

Median String Search: amélioration du temps de calcul

```
1.  MedianStringSearch (DNA, t, n, l)
2.  bestWord ← AAA…A
3.  bestDistance ← ∞
4.      for each l-mer s from AAA…A to TTT…T
          if TotalDistance(s,DNA) < bestDistance
5.          bestDistance←TotalDistance(s,DNA)
6.          bestWord ← s
7.  return bestWord
```

Structurer la recherche

Pour le problème Median String Search on a besoin de considérer tous les 4^l l-mers possibles:

l

aa… aa
aa… ac
aa… ag
aa… at
.
.
tt… tt

Comment organiser cette recherche?

Représentation alternative de l'espace de recherche

- Let **A** = 1, **C** = 2, **G** = 3, **T** = 4
- Alors les séquences de AA…A a TT…T deviennent:

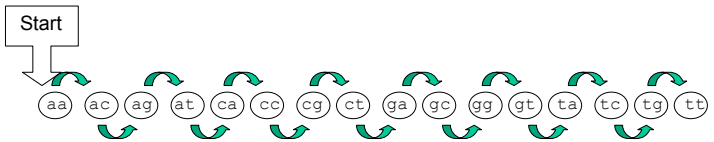
l

11…11
11…12
11…13
11…14
.
.
44…44

- Note que les séquences ci-dessus simplement liste tous les nombres comme si on était en train de compter en base 4 sans utiliser le 0.

Liste chaînée

- Supposons $l = 2$

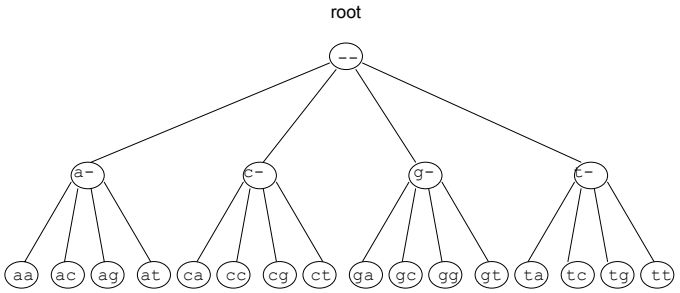


- On visitera tous les prédécesseurs d'une séquence avant de visiter la séquence elle même

Listes chaînées (continuation)

- La liste chaînée n'est pas la meilleure structure de données pour trouver les motifs
- On groupe les séquences par leur préfixes

Arbre de recherche



Analyse de l'arbre de recherche

- Caractéristiques de l'arbre de recherche:
  - Les séquences correspondent aux feuilles
  - Le père de chaque noeud est le préfixe de ses fils
- Comment peut-on "bouger" sur cet arbre ?

Bouger sur l'arbre de recherche

- On défini 4 façons possibles de se déplacer sur l'arbre de recherche :
  - Se déplacer sur la prochaine feuille
  - Visiter toutes les feuilles
  - Visiter le prochain noeud
  - Ignorer les fils d'un noeud

Visiter la prochaine feuille

Etant donnée une feuille **a** , nous devons calculer la "prochaine" feuille a visiter:

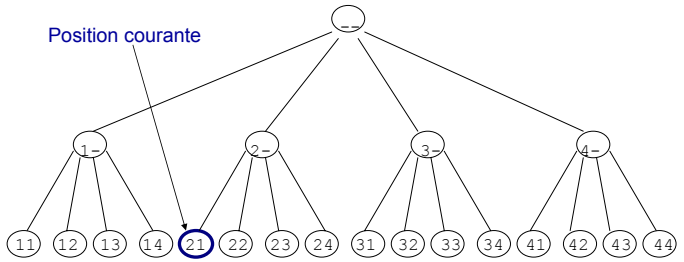
```
1.  NextLeaf( a,L, k )      // a : tableau de digits
2.  for i ← L to 1          // L: longueur du tableau
3.     if a_i < k           // k : valeur max des digits
4.        a_i ← a_i + 1
5.           return a
6.        a_i ← 1
7.  return a
```

Visite de la prochaine feuille (continuation)

- L'algorithme incrément le digit moins significatif comme dans le calcul en base k usuel
- et il "ajoute 1" à la prochaine position quand le digit a rejoint sa valeur maximale

NextLeaf: exemple

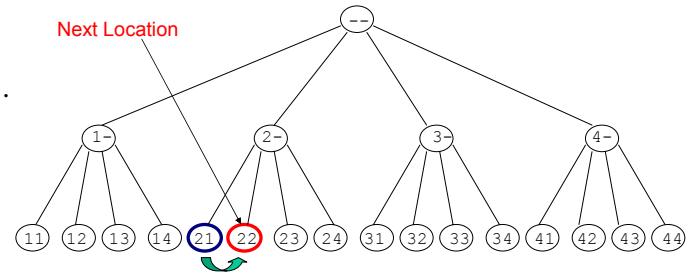
- Bouger sur la prochaine feuille:



NextLeaf: Exemple (continuation)

- Bouger sur la prochaine feuille:

.



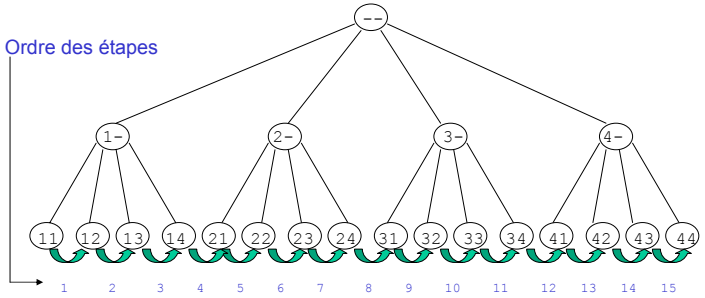
Visiter toutes les feuilles

- générer toutes les permutations en ordre croissant :

```
1.   AllLeaves(L,k)   // L: longueur de la sequence
2.   a ← (1,...,1)    // k : valeur max des digits
3.   while forever    // a : tableau de digits
4.      output a
5.      a ← NextLeaf(a,L,k)
6.      if a = (1,...,1)
7.         return
```

Visiter toutes les feuilles: exemple

- Bouger sur toutes les feuilles :



Recherche en profondeur

Comment peut-on rechercher tous les nœuds de l'arbre ?

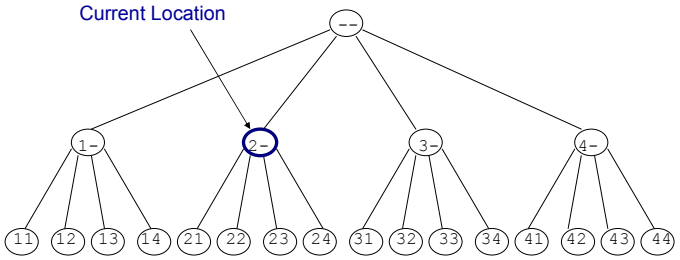
On adopte le parcours en profondeur

Visiter le prochain noeud

```
1.  NextVertex(a,i,L,k)      // a : tableau de digits
2.    if i < L               // i : longueur du prefix
3.      a i+1 ← 1            // L: longueur max
4.      return ( a,i+ 1)     // k : valeur max des digits
5.    else
6.      for j ← L to 1
7.        if a_j < k
8.          a_j ← a_j + 1
9.          return( a,j )
10.   return(a,0)
```

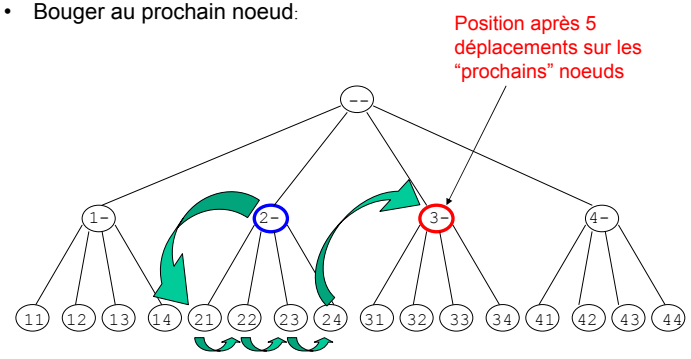

Exemple

- Bouger au prochain noeud:



Exemple

- Bouger au prochain noeud:



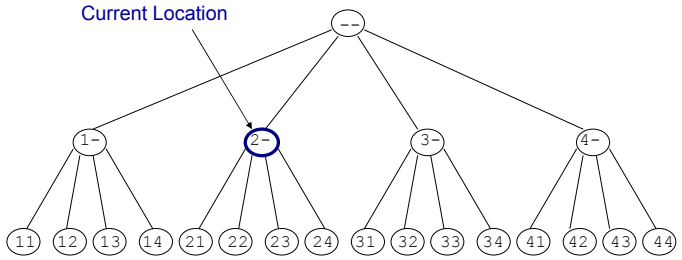
Déplacement en ignorant un noeud

- Etant donné un préfixe (noeud interne), trouver le prochain noeud après avoir ignoré tous ses fils

```
1.    Bypass(a,i,L,k)     // a: tableau de digits
2.    for j ← i to 1      // i: longueur du prefix
3.       if a_j < k       // L: longueur max
4.          a_j ← a_j + 1 // k: valeur max des digits
5.          return(a,j)
6.    return(a,0)
```

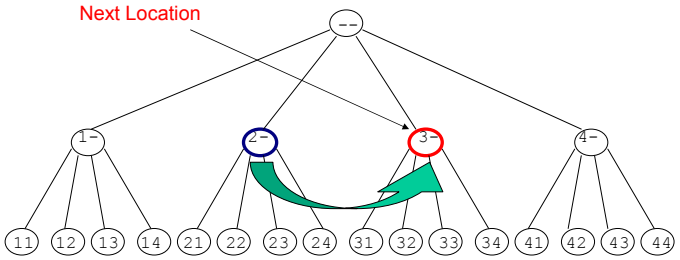

Deplacement en ignorant un noeud : exemple

- Ignorer les descendants de "2-":



Exemple

- Ignorer les descendants de "2-":



On regarde de nouveau l'algorithme BruteForceSearch: BruteForceSearchAgain

```
1.   BruteForceMotifSearchAgain(DNA, t, n, ℓ)
2.   s ← (1,1,…, 1)
3.   bestScore ← Score(s,DNA)
4.   while forever
5.         s ← NextLeaf (s, t, n– ℓ + 1)
6.         if (Score(s,DNA) > bestScore)
7.               bestScore ← Score(s, DNA)
8.               bestMotif ← (s1,s2 , . . . , st)
9.   return bestMotif
```

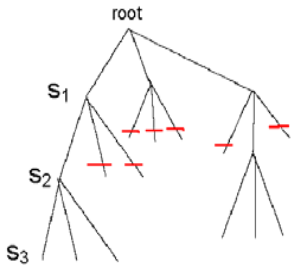

Est-ce qu'on peut faire mieux ?

- Les ensembles de **s=**(s_1, s_2, …,s_t) peuvent avoir un profile faible pour les premières *i* positions (s_1, s_2, …,s_i)
- Chaque ligne de l'alignement peut ajouter au plus ***l*** au Score
- Situation optimiste: si toutes les restantes *(t-i)* positions (s_{i+1}, …s_t) ajoute

  $(t - i) * l$ to Score(**s**,*i*,***DNA***)
- si Score(**s**,*i*,***DNA***) + $(t - i) * l$ < **BestScore**, il n'y a pas de raison pour chercher dans les noeuds du sous-arbre courant
  - Utiliser **ByPass()**

Un algorithme Branch and Bound pour la recherche des motifs

- Comme chaque niveau de l'arbre demande de descendre dans la recherche, décharger un préfixe signifie décharger toutes ses sous-branches
- Cela nous permet d'éviter de regarder a $(n - l + 1)^{t-i}$ feuilles
  - Utiliser **NextVertex()** et **ByPass()** pour le deplacement dans l'arbre



Pseudocode pour la recherche par Branch and Bound

```
1.   BranchAndBoundMotifSearch(DNA,t,n,ℓ)
2.   s ← (1,…,1)
3.   bestScore ← 0
4.   i ← 1
5.   while i > 0
6.       if i < t
7.             optimisticScore ← Score(s, i, DNA) +(t – i ) * ℓ
8.             if optimisticScore < bestScore
9.               (s, i) ← Bypass(s,i, n-ℓ+1)
10.            else
11.              (s, i) ← NextVertex(s, i, n-ℓ+1)
12.      else
13.            if Score(s,DNA) > bestScore
14.              bestScore ← Score(s)
15.              bestMotif ← (s1, s2, s3, …, st)
16.           (s,i) ← NextVertex(s,i,t,n-ℓ+ 1)
17.  return bestMotif
```

Amelioration de l'algorithme de recherche de la chaine mediane

On veut utiliser l'algorithme "median string search" mais avec l'idee du Branch and Bound !

- Il faut noter que si la distance totale d'un prefixe est plus grande que celle du meilleur mot obtenu jusqu'a la :

  TotalDistance (***prefix***, ***DNA***) > ***BestDistance***

  alors il est inutile d'explorer le reste du mot.

- On peut alors eliminer cette branche (avec BYPASS) du parcours d'exploration.

Bounded Median String Search

```
1.  BranchAndBoundMedianStringSearch(DNA,t,n,l)
2.  s ← (1,...,1)
3.  bestDistance ← ∞
4.  i ← 1
5.  while i > 0
6.    if i < l
7.      prefix ← string corresponding to the first i nucleotides of s
8.      optimisticDistance ← TotalDistance(prefix,DNA)
9.      if optimisticDistance > bestDistance
10.                (s, i ) ← Bypass(s,i, l, 4)
11.     else
12.                (s, i ) ← NextVertex(s, i, l, 4)
13.   else
14.     word ← nucleotide string corresponding to s
15.     if TotalDistance(s,DNA) < bestDistance
16.               bestDistance ← TotalDistance(word, DNA)
17.               bestWord ← word
18.     (s,i ) ← NextVertex(s,i,l, 4)
19. return bestWord
```


Amélioration des bornes

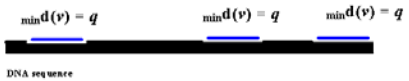
- Etant donnée un ***l***-mer ***w***, diviser-le en deux parties à la position ***i***
  - ***u*** : préfixe w_1, …, w_i,
  - ***v*** : suffixe w_{i+1}, ..., w_l
- Trouver la distance minimale pour ***u*** dans une séquence
- aucune instance de ***u*** dans la séquence a une distance plus petite que la distance minimale
- Noter que cela ne dit rien sur le fait que ***u*** fait partie ou pas d'un motif. On obtient seulement une distance minimale pour le préfixe ***u***

Amélioration des bornes (continuation)

- En répétant le processus pour un suffixe ***v***, il nous donne une distance minimale pour ***v***
- Comme ***u*** et ***v*** sont deux sous-chaînes de ***w***, qui sont incluses dans le motif ***w***, on peut faire l'hypothèse que la distance minimale de ***u*** plus la distance minimale de ***v*** peut seulement être plus petite que la distance minimale pour ***w***

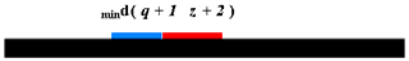
Meilleures bornes

Searching for prefix V
We may find many instances of prefix V with a minimum distance q



Likewise for U

But for U and V combined, U is not at its minimum distance location, neither is V



But at least we know w (prefix u suffix v) cannot have distance *less* than $_{min}d(v) + {}_{min}d(u)$

Meilleures bornes (continuation)

- si d(***prefix***) + d(***suffix***) ≥ ***bestDistance***:
  - Le motif ***w*** (*prefix.suffix)* ne peut pas donner un meilleur (plus petit) score que d(***prefix***) + d(***suffix***)
  - Dans ce cas, nous pouvons faire un **ByPass()**

Better Bounded Median String Search

```
1.  ImprovedBranchAndBoundMedianString(DNA,t,n,ℓ)
2.    s = (1, 1, …, 1)
3.    bestdistance = ∞
4.    i = 1
5.    while i > 0
6.        if i < ℓ
7.          prefix = nucleotide string corresponding to (s1, s2, s3, …, si )
8.          optimisticPrefixDistance = TotalDistance (prefix, DNA)
9.            if (optimisticPrefixDistance < bestsubstring[ i ])
10.             bestsubstring[ i ] = optimisticPrefixDistance
11.             if (ℓ- i < i )
12.               optimisticSufxDistance = bestsubstring[ℓ-i ]
13.             else
14.               optimisticSufxDistance = 0;
15.             if optimisticPrefixDistance + optimisticSufxDistance ≥ bestDistance
16.               (s, i ) = Bypass(s, i, ℓ 4)
17.             else
18.               (s, i ) = NextVertex(s, i, ℓ4)
19.       else
20.         word = nucleotide string corresponding to (s1,s2, s3, …, si)
21.         if TotalDistance( word, DNA) < bestDistance
22.           bestDistance = TotalDistance(word, DNA)
23.           bestWord = word
24.           (s,i) = NextVertex(s, i,ℓ 4)
25.   return bestWord
```


Encore quelques considérations sur le problème de la recherche des Motifs

- Motif Finding et Median String Search sont des **algorithmes exactes.**
- Ils trouvent toujours la solution optimale, et ils peuvent être très lents dans la pratique.
- Plusieurs algorithmes ont été conçus pour obtenir des solutions sous-optimales mais avec un temps d'exécution plus efficace.

Some Motif Finding Programs

- **CONSENSUS**
  Hertz, Stormo (1989)
- **GibbsDNA**
  Lawrence et al (1993)
- **MEME**
  *Bailey, Elkan (*1995)
- **RandomProjections**
  Buhler, Tompa (2002)
- **MULTIPROFILER** *Keich, Pevzner (2002)*
- **MITRA**
  Eskin, Pevzner (2002)
- **Pattern Branching**
  Price, Pevzner (2003)
- **SPATT**
  Nuel (2003)

CONSENSUS: Greedy Motif Search

- CONSENSUS trouve les 2 l-mers les plus proches dans les séquences 1 et 2 et forme une *matrice d'alignement 2 x l* avec *Score(***s***,2,DNA)*
- Pour chacune des *t-2* itérations suivante CONSENSUS trouve un "meilleur" *l*-mer dans la séquence *i* a partir de la matrice d'alignement *(i-1) x l* déjà construite pour les premiers *(i-1)* séquences
- cad, il trouve un *l*-mer dans la séquence *i* qui maximise

  *Score(***s***,i,DNA)*

  sous l'hypothèse que les premiers *(i-1) l*-mers ont été déjà choisis.
- CONSENSUS sacrifie la solution optimale pour la rapidité : en fait la plupart du temps est dépensée pour localiser les premiers 2 *l*-mers

« Motif challenge problem »

- Input:
  - *n* séquences de longueur *m* chacune.
- Output:
  - Motif *M*, de longueur *l*
  - Variantes intéressantes ont une distance de Hamming *d* de *M*

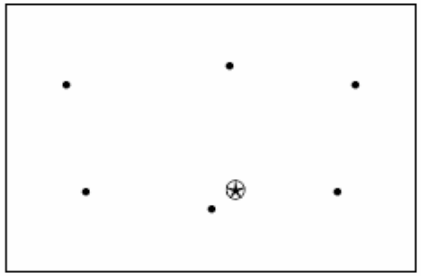
Comment procéder?

- Recherche exhaustive ?

CGAGTCACTGAGCTGTGCCAGAGCCCCAAAAAGTGGCTGCTAAAGT
GTTGGGTGTTCTCTCAAAATGATGACGAAGCTGGGTCTGAGACAGA
AGTGTCCTGCTATAATTTAACTGATTTGAACCGCAACACTTCCGAA
GGGGATCGGATCCCATGCGCTGAGTTAGGACTCCACAGTCAGAGAC
AAGCAAACCATTTTCTATCGGAGCCCCGGCCTTAACCCCACGATTC
ATGTGAAAGTCCATTTTTCGTATCAGACGAGATGTGAGCATTTAGC
TGCTAGGATCAGAGTCAGAGTGACACTTAGTCAGAATGGGTCCCTG
GTTGCGACCACTTCCGAGGACCTTAAGACCTGAGCATAACGACTAC

- Le temps d'exécution est élevé

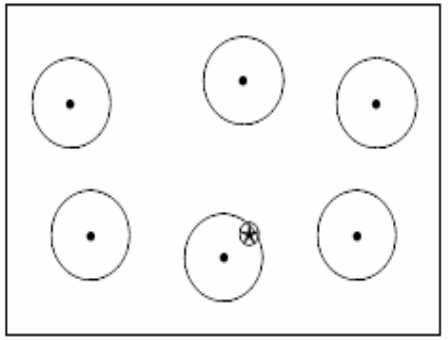
Comment rechercher dans **l'espace des motifs**?



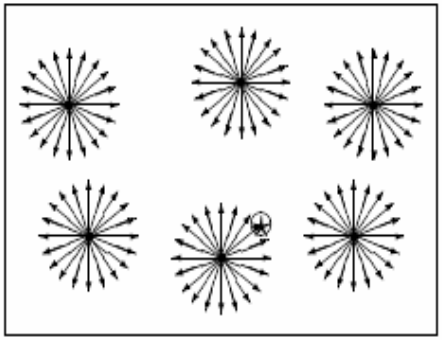
Commencer par des chaînes de caractères aléatoires

Recherche l'espace pour trouver l'étoile.

Recherche dans des voisinages suffisamment petits

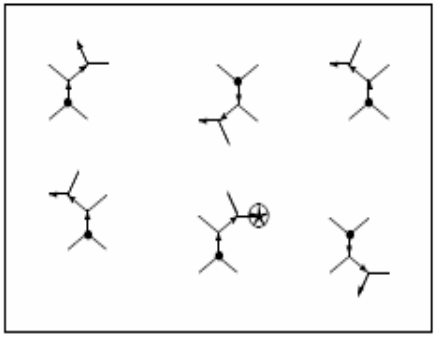


Recherche locale exhaustive



Plein de travail, pour la plupart non nécessaire

Meilleur voisin



S'éloigner des chaînes de départ ("seeds").

Trouver le meilleur voisin – ayant le meilleur score.

Ne considérer pas les chemins où la borne supérieure n'est pas au moins bonne comme le meilleur score obtenu jusqu'a la.

Scoring

- PatternBranching utilise un score de distance totale :

  pour chaque séquence S_i dans l'échantillon de départ $S = \{S_1, \ldots, S_n\}$, soit

  $$d(A, S_i) = \min\{d(A, P) \mid P \in S_i\}.$$

  Alors la distance totale de A de l'echantillon

  $$d(A, S) = \sum_{S_i \in S} d(A, S_i).$$

- Pour un motif *A*, soit *D=Neighbor(A)* l'ensemble des motifs qui diffèrent de *A* en exactement 1 position.
- on définie *BestNeighbor(A)* comme le motif $B \in D=Neighbor(A)$ avec la distance totale minimale *d(B, S)*.

PatternBranching Algorithm

PatternBranching$(\mathcal{S}, l, k)$
$d \leftarrow \infty$
For each l-mer A_0 in $\mathcal{S}$
  For $j \leftarrow 0$ to k
    If $d(A_j, \mathcal{S}) < d$
      $Motif \leftarrow A_j$
      $d \leftarrow d(A_j, \mathcal{S})$
    $A_{j+1} \leftarrow \text{BestNeighbor}(A_j)$
Output *Motif*

PatternBranching Performance

- PatternBranching est plus rapide que d'autres algorithmes basés sur les motifs.
- Motif Challenge Problem:
  - Echantillon de n = 20 séquences
  - N = 600nt en longueur
  - motifs de longueur l = 15
  - k = 4 mutations

Algorithm	Success Rate	Running Time
PROJECTION	about 100%	2 minutes
MITRA	100%	5 minutes
MULTIPROFILER	99.7%	1 minute
PatternBranching	99.7%	3 seconds

PMS (Planted Motif Search)

- Génère tous les possibles *l*-mers a partir de la séquence d'entrée S_i. Soit C_i la collection de ces *l*-mers.
- Exemple:
  AAGTCAGGAGT
  C_i = 3-mers:
  AAG AGT GTC TCA CAG AGG GGA GAG AGT

All patterns at Hamming distance $d = 1$

AAG	**AGT**	**GTC**	**TCA**	**CAG**	**AGG**	**GGA**	**GAG**	**AGT**
CAG	CGT	ATC	ACA	AAG	CGG	AGA	AAG	CGT
GAG	GGT	CTC	CCA	GAG	TGG	CGA	CAG	GGT
TAG	TGT	TTC	GCA	TAG	GGG	TGA	TAG	TGT
ACG	ACT	GAC	TAA	CCG	ACG	GAA	GCG	ACT
AGG	ATT	GCC	TGA	CGG	ATG	GCA	GGG	ATT
ATG	AAT	GGC	TTA	CTG	AAG	GTA	GTG	AAT
AAC	AGA	GTA	TCC	CAA	AGA	GGC	GAA	AGA
AAA	AGC	GTG	TCG	CAC	AGT	GGG	GAC	AGC
AAT	AGG	GTT	TCT	CAT	AGC	GGT	GAT	AGG

Ordonnencement des listes

AAG	**AGT**	**GTC**	**TCA**	**CAG**	**AGG**	**GGA**	**GAG**	**AGT**
AAA	AAT	ATC	ACA	AAG	AAG	AGA	AAG	AAT
AAC	ACT	CTC	CCA	CAA	ACG	CGA	CAG	ACT
AAT	AGA	GAC	GCA	CAC	AGA	GAA	GAA	AGA
ACG	AGC	GCC	TAA	CAT	AGC	GCA	GAC	AGC
AGG	AGG	GGC	TCC	CCG	AGT	GGC	GAT	AGG
ATG	ATT	GTA	TCG	CGG	ATG	GGG	GCG	ATT
CAG	CGT	GTG	TCT	CTG	CGG	GGT	GGG	CGT
GAG	GGT	GTT	TGA	GAG	GGG	GTA	GTG	GGT
TAG	TGT	TTC	TTA	TAG	TGG	TGA	TAG	TGT

Elimination des copies

AAG	**AGT**	**GTC**	**TCA**	**CAG**	**AGG**	**GGA**	**GAG**	**AGT**
AAA	AAT	ATC	ACA	AAG	AAG	AGA	AAG	AAT
AAC	ACT	CTC	CCA	CAA	ACG	CGA	CAG	ACT
AAT	AGA	GAC	GCA	CAC	AGA	GAA	GAA	AGA
ACG	AGC	GCC	TAA	CAT	AGC	GCA	GAC	AGC
AGG	AGG	GGC	TCC	CCG	AGT	GGC	GAT	AGG
ATG	ATT	GTA	TCG	CGG	ATG	GGG	GCG	ATT
CAG	CGT	GTG	TCT	CTG	CGG	GGT	GGG	CGT
GAG	GGT	GTT	TGA	GAG	GGG	GTA	GTG	GGT
TAG	TGT	TTC	TTA	TAG	TGG	TGA	TAG	TGT

Trouver un motif commun a toutes les listes

- Suivre la procédure pour toutes les séquences
- Trouver le motif « commun » à toutes les L_i (une fois que les copies ont été éliminées)
- Ceci est le motif cherché

Temps d'exécution de PMS

- Il prends un temps pour
  - Générer les variantes $O\left(m\binom{l}{d}3^d\right)$
  - Ordonner les listes
  - Trouver et éliminer les copies
- la complexité en temps de l'algorithme:

$$O\left(nm\binom{l}{d}3^d\frac{l}{w}\right)$$

w est la longueur d'un mot dans l'ordinateur

Bibliographie

N.Jones and P.Pevzner, *An introduction to Bioinformatics Algorithms*, MIT press, 2004.

CONSENSUS
G.Z. Hertz and G.D. Stormo. Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps. In: Proceedings of the Third International Conference on Bioinformatics and Genome Research (H.A. Lim, and C.R. Cantor, editors). World Scientific Publishing Co., Ltd. Singapore, 1995. pages 201--216.

PatternBranching
A.Price, S.Ramabhadran, and P.Pevzner. Finding subtle motifs by branching from sample strings. In *Proceedings of the Second European Conference on Computational Biology (ECCB-03)*, pages 69–76, September 2003.