

Arbre des suffixes

et

Recherche de motifs

Arbre des suffixes

- ▷ Structure de données qui permet de représenter exactement tous les facteurs d'un texte
- ▷ Une feuille de l'arbre = un suffixe du mot

un facteur = un préfixe d'un suffixe

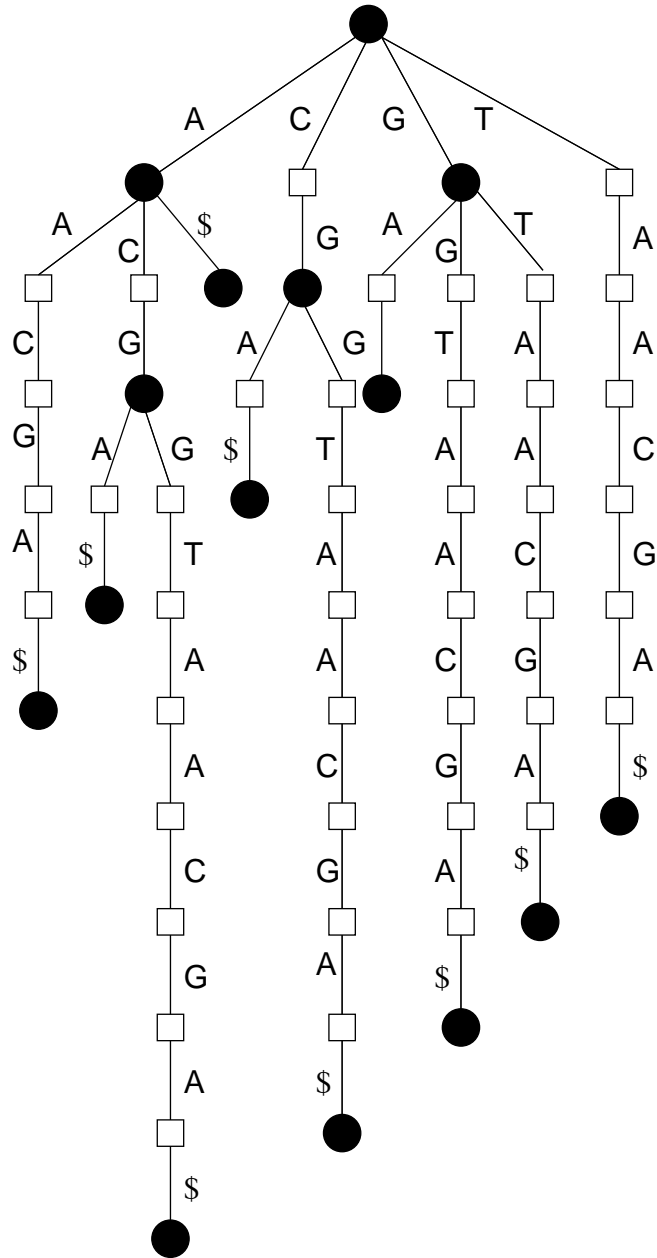


un chemin dans l'arbre = un facteur

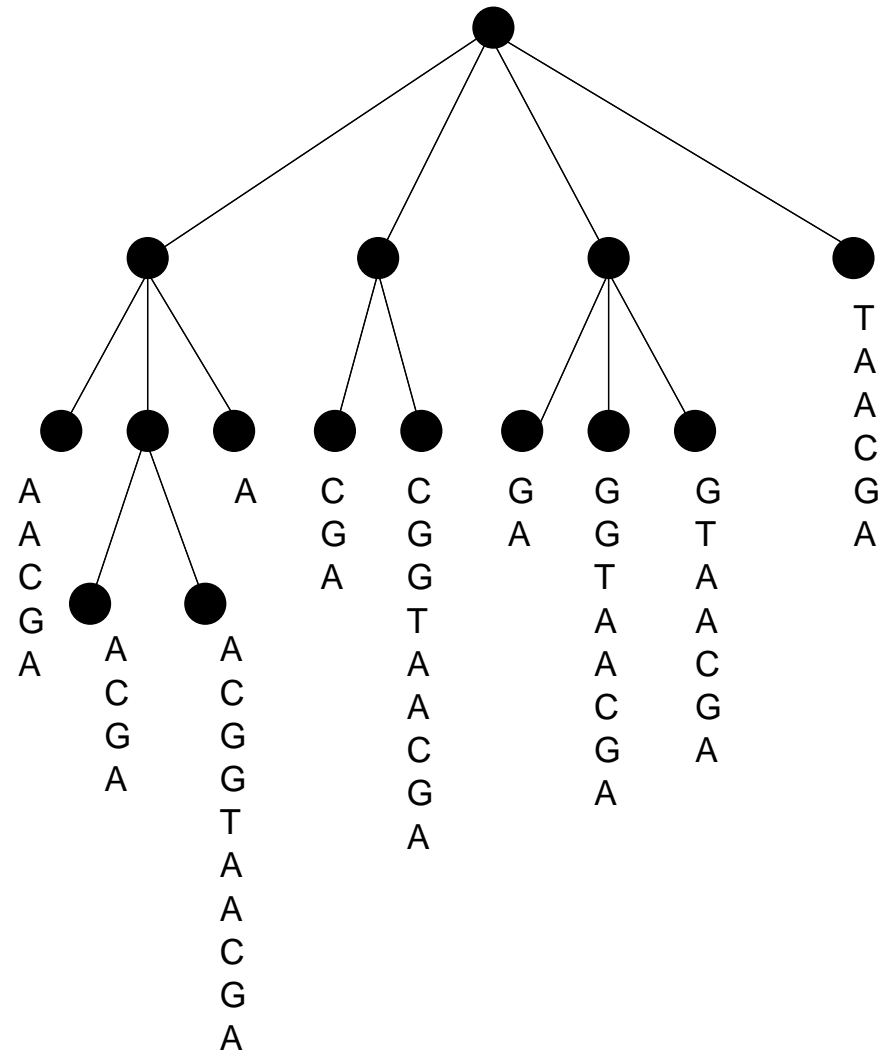
- ▷ Taille linéaire

Chaque nœud interne a au moins deux fils

Deuxième tentative : ACGGTAACGA\$ (\$: nouveau caractère)



une feuille = un suffixe



Arbre des suffixes de ACGGTAACGA

Compaction : on obtient une taille linéaire

Algorithmes de construction

Temps et espace linéaire

▷ **Weiner** (1973)

Algorithme historique

▷ **Mac Creight** (1976)

Construction à l'aide de liens suffixes

▷ **Ukkonen** (1995)

Algorithme on-line

Arbre des suffixes généralisés

▷ Un ensemble s_1, \dots, s_n de séquences

▷ **Arbre des suffixes généralisé**

un chemin de la racine à une feuille = un suffixe d'une séquence

▷ Construction incrémentale

- Arbre des suffixes pour s_1
- Ajout des autres séquences s_2, s_3, \dots

Espace et temps linéaire

Que peut-on faire avec un arbre des suffixes ?

- ▷ Recherche de motifs exacts
- ▷ Recherche de motifs avec k erreurs
- ▷ Recherche de répétitions
- ▷ Recherche du plus long facteur commun entre n séquences
- ▷ Séquences cycliques
- ▷ ...

Application : MUMmer

Alignement de génomes

Deux génomes A et B

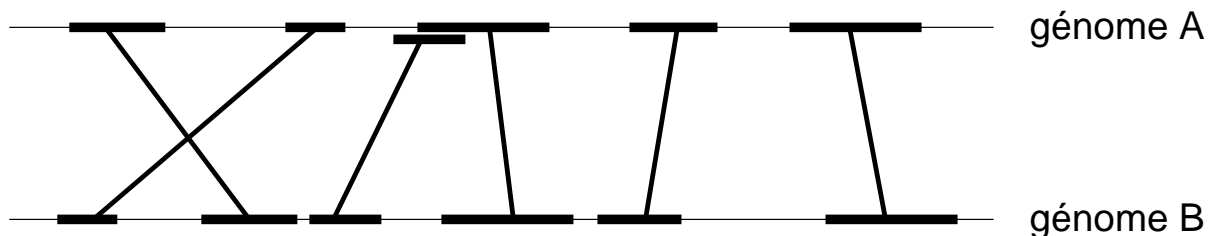
Etape 1 : *Détection des segments communs entre les deux séquences*

▷ construction de l'arbre des suffixes généralisé des deux séquences

▷ recherche des **MUM** (Maximal Unique Match)

unique : nœud de l'arbre avec exactement deux fils,
un pour chaque génome

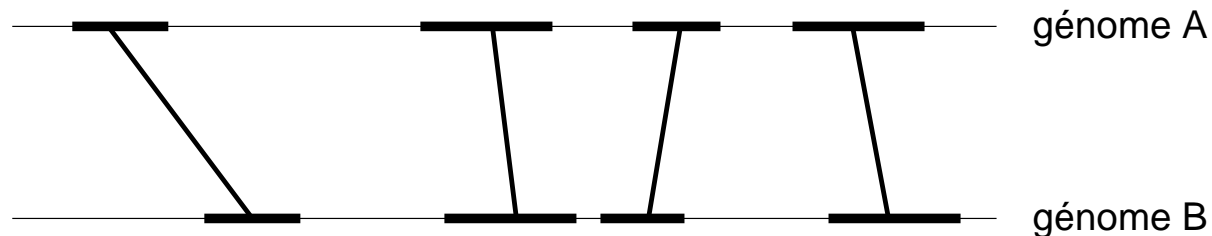
maximal : au cours de la construction de l'arbre



Etape 2 : *Extraction d'un ensemble cohérent de MUMs*

- ▷ Tris de MUMs suivant leurs coordonnées dans le génome *A*
- ▷ Extraction d'une sous-suite croissante pour le génome *B*

Utilise l'algorithme de la plus longue sous-séquence croissante, en pondérant chaque MUM par sa longueur, et en traitant les chevauchements.



Etape 3: *Enrichissement de l'alignement*

À partir de l'arbre des suffixes

- ▷ Identification des SNP (Single Nucleotide Polymorphism)
- ▷ Identification des répétitions en tandem
- ▷ Traitement des insertions : véritable insertion ou transposition ?
- ▷ Traitement des régions polymorphes (zones entre deux MUM sur les deux séquences)
 - région courte : alignement local de Smith&Waterman
 - région longue : recherche de MUM plus courts

Recherche de motifs exacts

Données :

un texte T de longueur n : $T(1..n)$

un motif M de longueur m : $M(1..m)$

Problème : Trouver toutes les occurrences de M dans T

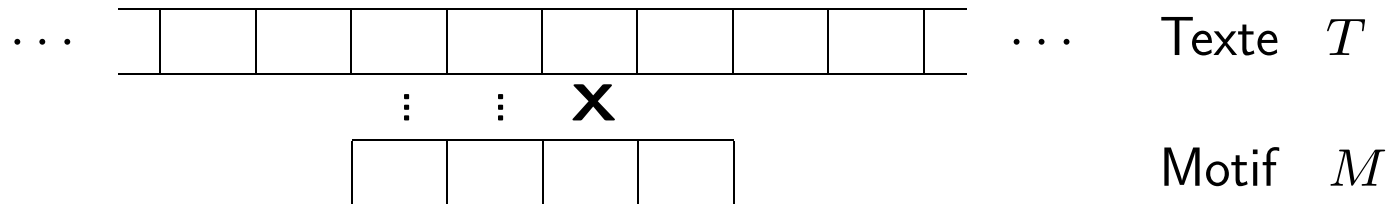
Des dizaines, (des centaines ?) d'algorithmes ...

- ▷ Méthodes comparatives : Algorithme *brute force* (immémorial), Boyer-Moore (1977)
- ▷ Méthodes numériques : Shift-Or (Baeza-Yates et Gonnet 1992), Karp et Rabin (1987)

Exemples d'application

- ▷ recherche de sites de clivage d'enzyme de restriction
- ▷ recherche de sites d'hybridation d'amorces
- ▷ localisation d'EST

Algorithme brute force



- Le motif balaye tout le texte, du début vers la fin.

Décalage = +1

- À chaque position du texte, on compare le texte au motif, jusqu'à rencontrer un mismatch, ou jusqu'à épuiser le motif.

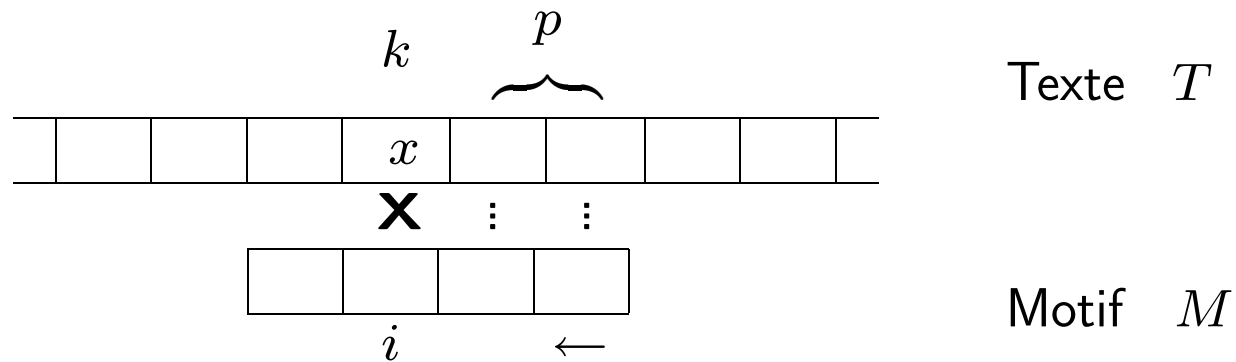
Complexité : $O(nm)$

Comment améliorer l'algorithme *brute force* ?

- ▷ On cherche à faire des décalages de plus de une position
- ▷ Bien sûr, ces décalages plus importants doivent rester corrects : ils ne doivent pas permettre de rater une occurrence du motif
- ▷ C'est possible en faisant un **pré-traitement** du motif M

Algorithme de Boyer-Moore

Le motif est lu de la fin vers le début. Le décalage a toujours lieu du début vers la fin du texte.



Trois informations sont exploitées :

- $p = M(i + 1..m)$
 - $T(k) \neq M(i)$
 - $T(k) = x$
- } règle du bon suffixe
- } règle du mauvais caractère

Règle du bon suffixe

Décaler le motif vers la droite de telle sorte que la prochaine occurrence du suffixe p dans M soit alignée avec la position k de T , et que le caractère suivant soit différent de $M(i)$.

GS(i) (good suffix) :

plus petit entier non nul l tel que le décalage de l positions soit pertinent :

- $M(i + 1 - l..m - l) = M(i + 1..m)$
- $M(i - l) \neq M(i)$

ou si l n'existe pas :

$l = m - k$, où k est le plus grand entier tel que $M(m - k + 1..m) = M(1..k)$

Exemples :

le motif *tactaga*

	i	1	2	3	4	5	6	7
GS(i)		7	7	7	7	7	2	1

le motif *gcagagag*

	i	1	2	3	4	5	6	7	8
GS(i)		7	7	7	2	7	4	7	1

Règle du mauvais caractère

Décaler le motif de telle sorte que la position k du texte soit alignée avec un caractère $T(k)$ du motif.

$\text{BC}(x)$ (bad character) :

longueur du plus long suffixe de M qui ne contient pas x , sauf éventuellement en dernière position

Exemple : le motif *gcagagag*

<i>a</i>	<i>c</i>	<i>g</i>	<i>t</i>
1	6	2	8

Quand un mismatch intervient entre la position i du motif et la position k du texte, le décalage se fait de $\max\{\text{GS}(i), \text{BC}(T(k)) - (m - i)\}$ positions.

Points forts

- ▷ **Complexité** : $O(mn)$ dans le pire des cas, mais **sous-linéaire en moyenne**

Sous-linéaire: un caractère est lu moins d'une fois.

- ▷ Terriblement efficace en pratique si l'alphabet est grand, ou si le motif est long

Points faibles

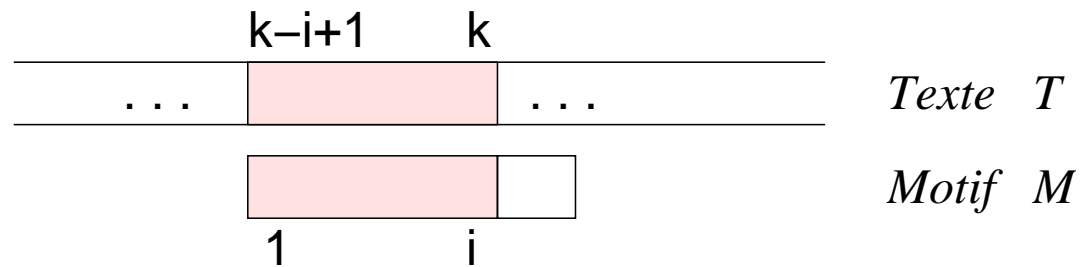
- ▷ Pré-traitement en temps linéaire, mais **délicat à implémenter** (non détaillé)
- ▷ Moins convaincant pour de courts motifs sur un petit alphabet (l'ADN par exemple)

Algorithme Shift-Or

Décalage de bits

\mathcal{B} : matrice $m \times n$ sur 0 et 1

$$\mathcal{B}(i, k) = 1 \Leftrightarrow M(1..i) = T(k - i + 1..k)$$



Les occurrences du motif M se trouvent aux positions p telles que

$$\mathcal{B}(m, p + m - 1) = 1.$$

Calcul de la matrice \mathcal{B}

Exemple : le motif *tactaga*

On définit quatre vecteurs (un par lettre de l'alphabet)

$U(a)$	$U(c)$	$U(g)$	$U(t)$
0	0	0	1
1	0	0	0
0	1	0	0
0	0	0	1
1	0	0	0
0	0	1	0
1	0	0	0

$$U(x)(i) = 1 \iff M(i) = x$$

Première colonne de \mathcal{B}

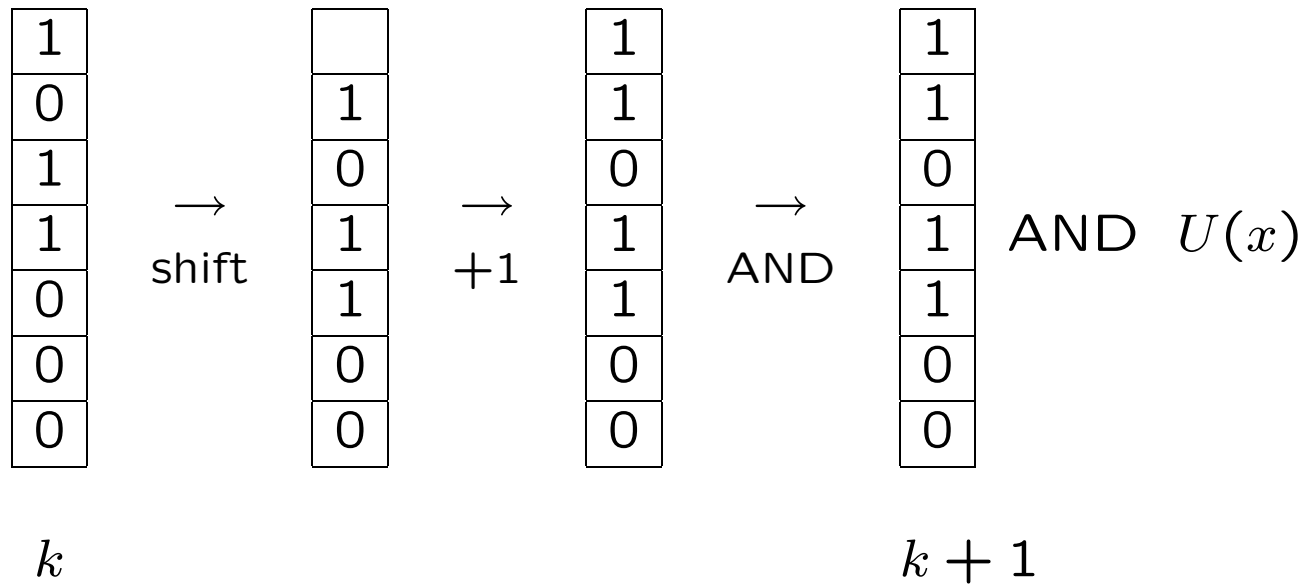
0
0
0
0
0
0
0

si $T(0) \neq M(0)$

1
0
0
0
0
0
0

si $T(0) = M(0)$

Colonnes suivantes : la colonne $k + 1$ à partir de la colonne k



où $x = T(k + 1)$.

AND est l'opérateur qui effectue le **Et logique** bit par bit.

Points forts

- ▷ Pré-traitement quasi inexistant
- ▷ Facile à implémenter
- ▷ Fonctionne bien si le motif est de taille inférieure à un mot machine: les opérations se font alors en temps constants. On obtient alors une complexité linéaire.
- ▷ S'adapte facilement au cas de motifs approchés

Points faibles

- ▷ Inadapté quand le motif est long

Algorithme de Karp-Rabin

Arithmétique modulo et décalage

d : taille de l'alphabet

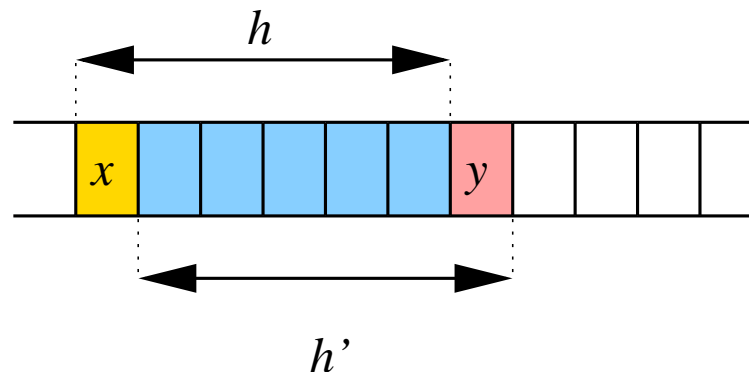
Un mot est codé par un entier en base d , grâce à une **fonction de hachage** h

$$h(u) = (u_0d^{m-1} + \dots + u_{m-1}) \text{ modulo } q$$

q est choisi comme le plus grand entier : le modulo se fait implicitement.

Déroulement de l'algorithme

1. Calcul de $h(M)$
2. Calcul de $h(T(0..m - 1))$
3. Déplacement d'une fenêtre de longueur m le long du texte, de 1 en 1, avec réactualisation de la valeur de h (Temps constant)



$$h' = (d(h - xd^{m-1}) + y) \text{ modulo } q$$

4. Quand on trouve une fenêtre f telle que $h(f) = h(u)$, on vérifie en comparant f et u caractère par caractère

Points forts

- ▷ Facile à implémenter
- ▷ Efficace en pratique
- ▷ Principe repris dans BLAST

Points faibles

- ▷ Complexité en $O(mn)$ dans le pire des cas
- ▷ Pas le plus efficace en pratique

Recherche d'oligonucléotides sur-représentés

Trouver les facteurs de transcription impliqués dans la régulation transcriptionnelle d'une famille de gènes

Exemple

▷ 12 gènes de la levure, régulés par la méthionine

SAM2, MET6, MIUP3, MET30, MET3, MET14, MET1, SAM1, MET17, ZWF1, MET2

▷ analyse de la région -800 -1

▷ modèle de fond : %GC, régions intergéniques de la levure

Van Helden, J., Andre, B., and Collado-Vides, J.: *Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotides frequencies.* Journal of Molecular Biology 281(5), 827-842, 1998

cacgtg	cacgtg cacgtg	0.000147	13	1.26	1.00e-9	2.1e-6
ccacag	ccacag ctgtgg	0.000259	11	2.22	2.10e-5	4.5e-2
acgtga	acgtga tcacgt	0.000358	13	3.1	2.20e-5	4.6e-2
aactgt	aactgt acagtt	0.000613	17	5.28	3.90e-5	8.0e-2
actgtg	actgtg cacagt	0.000366	12	3.16	0.00011	2.4e-1
gccaca	gccaca tgtggc	0.000299	10	2.59	0.00037	7.6e-1
gcttcc	gcttcc ggaagc	0.000416	12	6.6	0.00037	7.7e-1
séquence	identifiant	fréq. att.	n.o.	n.a.	P-value	E-value

- fréq. att.* : fréquence attendue du motif,
à partir d'un modèle constitué de régions inter-géniques
- n.o.* : nombre d'occurrences observé
- n.a.* : nombre d'occurrences attendu
- P-value* : probabilité du nombre d'occurrences observé
- E-value* : nombre de motifs attendu avec cette probabilité

Alignement des motifs trouvés

cluster # 1		cluster # 2	
tcacgt..	..acgtga	aactgt..	..acagtt
.cacgtg.	.cacgtg.	.actgtg.	.cacagt.
..acgtga	tcacgt..	..ctgtgg	ccacag..
tcacgtga	tcacgtga	aactgtgg	ccacagtt

complexe Met4p/Cbfl/Met28

Met31p