

# Séquences et architectures II

Mathieu Giraud  
giraud@lifl.fr

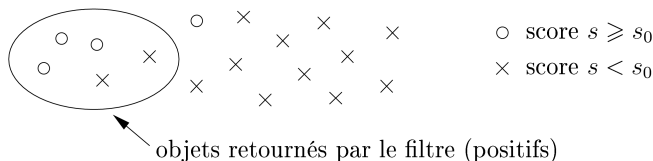
Équipe Bioinfo/Sequoia – LIFL – USTL

Master recherche informatique  
novembre 2006

- Algorithmes, mémoire et architecture
  - Localité mémoire
  - Algorithmes efficaces pour la mémoire...
  - ... ou du moins conscients de leurs limites
  - Circuits spécialisés, FPGA
- Architectures systoliques
  - Programmation dynamique
  - Fonctionnement systolique
  - Optimisations
- Heuristiques à base de graines
  - Heuristiques
  - Représentations d'ensembles, filtres Bloom
  - Amélioration des graines

# Pourquoi utiliser une heuristique ?

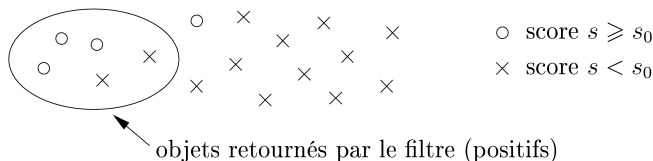
Un calcul d'alignement local est un **filtrage** retournant certaines positions dans la banque.



Vrais positifs  $T^{\oplus}$     Faux positifs  $F^{\oplus}$   
Faux négatifs  $F^{\ominus}$     Vrais négatifs  $T^{\ominus}$

# Pourquoi utiliser une heuristique ?

Un calcul d'alignement local est un **filtrage** retournant certaines positions dans la banque.

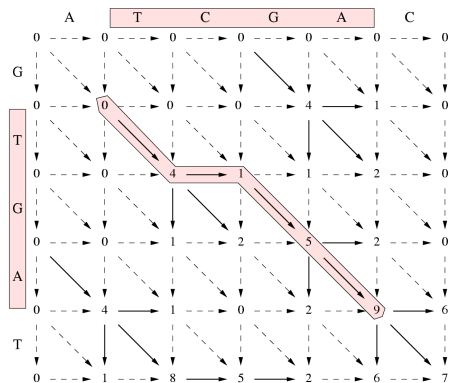


Vrais positifs  $T^{\oplus}$     Faux positifs  $F^{\oplus}$   
Faux négatifs  $F^{\ominus}$     Vrais négatifs  $T^{\ominus}$

$$\text{Selectivité } S_{\ell} = \frac{(T^{\oplus} + F^{\oplus})}{(T^{\oplus} + F^{\oplus} + T^{\ominus} + F^{\ominus})}$$

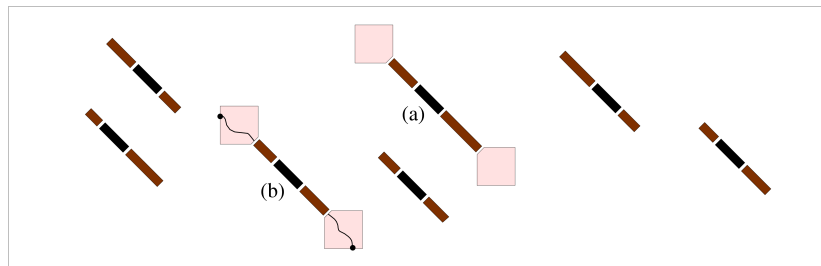
$$\text{Sensibilité } S_n = \frac{T^{\oplus}}{(T^{\oplus} + F^{\ominus})} \quad \text{Spécificité } S_p = \frac{T^{\ominus}}{(T^{\oplus} + F^{\oplus})}$$

# Heuristique à base de graines



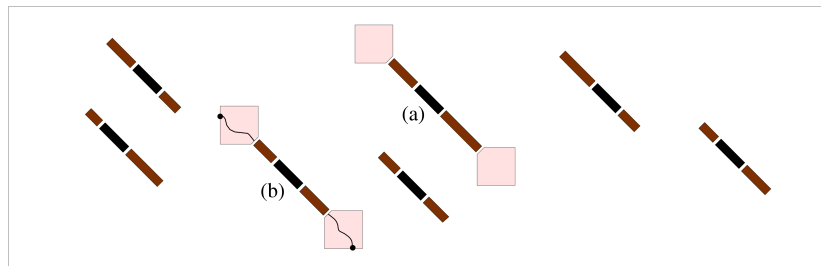
T C G A  
| - | |  
T - G A

# Heuristique à base de graines



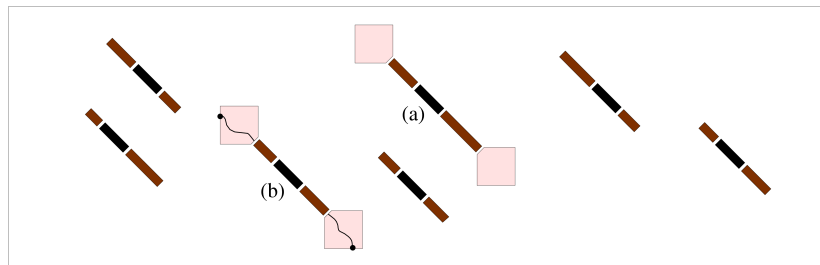
- 1. Localiser des *graines* de taille  $w = 11$  (80%)

# Heuristique à base de graines



- 1. Localiser des *graines* de taille  $w = 11$  (80%)
- 2. Étendre les graines avec une petite tolérance

# Heuristique à base de graines



- 1. Localiser des *graines* de taille  $w = 11$  (80%)
- 2. Étendre les graines avec une petite tolérance
- 3. Réaliser les calculs de programmation dynamique sur un nombre réduit de positions

# Indexation des graines

requête **GGTACCAAACCTCGGACCTGATACGTTC...**

# Indexation des graines

requête **GGTACCAAAC****TCGGAC**CCTGATACGTTC...

# Indexation des graines

requête **GGTACCAA**ACT**TCGGAC**CTGATACGTTC...

TCGCT	84
<b>TCGGA</b>	<b>154</b>
	<b>1238</b>
	<b>8470</b>
TCGGC	651

# Indexation des graines

requête **GGTACCAA**CTCGGACCTGATACGTTC...

**154** ...TTACTTCGGAGTCAA...

**1238** ...GATGATCGGAATTTC...

**8470** ...CATACTCGGAACTCA...

TCGCT 84  
TCGGA 154  
1238  
8470  
TCGGC 651



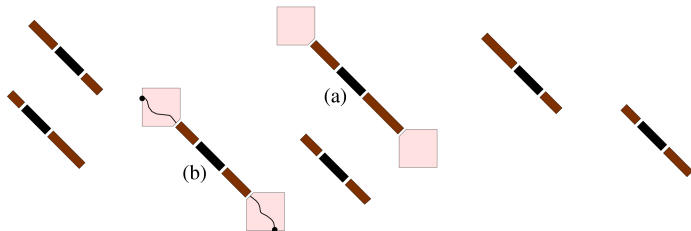
# Indexation des graines

	voisinage	graine	voisinage	
requête	GGTAC	CAA	CTCGG	ACCTGATACGTTC...
154	...	TTACTTCGG	GAGTCAA	... X
1238	...	GATGATCGG	AATTTTC	... X
8470	...	CATACTCGG	AACTCA	... ✓ puis étape 3

TCGCT	84
TCGGA	154
	1238
	8470
TCGGC	651

# Est-ce qu'un ordinateur fait encore des calculs ?

## Recherche de similarités entre $m$ et $n$



# Est-ce qu'un ordinateur fait encore des calculs ?

## Recherche de similarités entre $m$ et $n$

pour chaque graine  $s \in m$

**si**  $s$  apparaît dans  $n$

**alors**

soit  $v_m$  et  $v_n$  les voisinages de  $s$  dans  $m$  et  $n$

**si**  $v_m$  et  $v_n$  s'alignent sommairement

**alors**

soit  $V_m$  et  $V_n$  des voisinages plus étendus  
aligner  $V_m$  et  $V_n$

# Est-ce qu'un ordinateur fait encore des calculs ?

## Recherche de similarités entre $m$ et $n$

pour chaque graine  $s \in m$

**si**  $s$  apparaît dans  $n$

**alors**

soit  $v_m$  et  $v_n$  les voisinages de  $s$  dans  $m$  et  $n$

**si**  $v_m$  et  $v_n$  s'alignent sommairement

**alors**

soit  $V_m$  et  $V_n$  des voisinages plus étendus  
aligner  $V_m$  et  $V_n$

*parcours aléatoire de la séquence  $n$  ?*

# Est-ce qu'un ordinateur fait encore des calculs ?

## Recherche de similarités entre $m$ et $n$

pour chaque graine  $s \in m$

**si**  $s$  apparaît dans  $n$

**alors**

soit  $v_m$  et  $v_n$  les voisinages de  $s$  dans  $m$  et  $n$

**si**  $v_m$  et  $v_n$  s'alignent sommairement

**alors**

soit  $V_m$  et  $V_n$  des voisinages plus étendus  
aligner  $V_m$  et  $V_n$

*parcours aléatoire* de la séquence  $n$  ?

Étape 1 de  $\longrightarrow$  il faut déterminer si une graine de  $m$  (ACTGCCTAGCT) était présente ou non dans  $n$  ?

Comment stocker tous les mots de taille  $w = 11$  présents dans une requête ( $\times 1000 - \times 10^9$  caractères) ?

Est-ce qu'un élément appartient à un ensemble donné?

- dictionnaire des mots correctement orthographiés

Est-ce qu'un élément appartient à un ensemble donné ?

- dictionnaire des mots correctement orthographiés
- liste de morceaux disponibles d'un fichier échangé en P2P
- liste des URLs des pages conservées dans un cache web

Est-ce qu'un élément appartient à un ensemble donné?

- dictionnaire des mots correctement orthographiés
- liste de morceaux disponibles d'un fichier échangé en P2P
- liste des URLs des pages conservées dans un cache web
- ensemble des graines dans une requête ( $w$ -mots)

Dans un univers  $U$ , soit un ensemble  $S = \{x_1, \dots, x_n\} \subset U$ .  
Soit un élément  $y \in U$ . Appartient-il à  $S$  ?

- mémoire de  $|U|$  bits (1 bit par élément)

# Représentation d'ensembles

Dans un univers  $U$ , soit un ensemble  $S = \{x_1, \dots, x_n\} \subset U$ .  
Soit un élément  $y \in U$ . Appartient-il à  $S$  ?

- mémoire de  $|U|$  bits (1 bit par élément)
- mémoire d'environ  $n \cdot \log_2 |U|$  bits (liste des éléments)

On n'a qu'une mémoire trop petite, de taille  $M$ .  
Comment se souvenir de  $S$  ?

Peut-on tolérer un faible taux de faux positifs ?

- dictionnaire des mots correctement orthographiés
- liste de morceaux disponibles d'un fichier échangé en P2P
- liste des URLs des pages conservées dans un cache web
- ensemble des graines dans une requête ( $w$ -mots)

# Représentation d'ensembles avec faux positifs

Peut-on tolérer un faible taux de **faux positifs** ?

- dictionnaire des mots correctement orthographiés  
→ **on laisse passer quelques mots incorrects !**
- liste de morceaux disponibles d'un fichier échangé en P2P
- liste des URLs des pages conservées dans un cache web
- ensemble des graines dans une requête ( $w$ -mots)

# Représentation d'ensembles avec faux positifs

Peut-on tolérer un faible taux de faux positifs ?

- dictionnaire des mots correctement orthographiés  
→ on laisse passer quelques mots incorrects !
- liste de morceaux disponibles d'un fichier échangé en P2P  
→ on demande à tort un morceau de fichier
- liste des URLs des pages conservées dans un cache web  
→ on demande à tort une adresse que le cache n'a pas
- ensemble des graines dans une requête ( $w$ -mots)

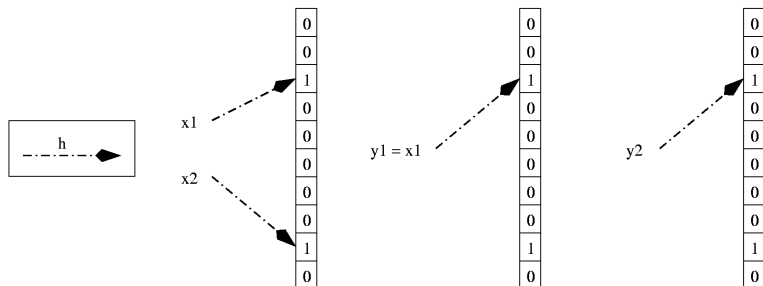
Peut-on tolérer un faible taux de **faux positifs** ?

- dictionnaire des mots correctement orthographiés  
→ on laisse passer quelques mots incorrects !
- liste de morceaux disponibles d'un fichier échangé en P2P  
→ on demande à tort un morceau de fichier
- liste des URLs des pages conservées dans un cache web  
→ on demande à tort une adresse que le cache n'a pas
- ensemble des graines dans une requête ( $w$ -mots)  
→ on envoie quelques faux positifs à l'extension

# Fonctions de hachage

Fonction  $h : U \longrightarrow [1 \dots M]$  avec  $M < |U|$ .

La fonction ventile "aléatoirement" les éléments de  $U$ .



$S = \{x_1, x_2\}$ ,  $M = 10$ ,  $h(x_1) = 3$ ,  $h(x_2) = 9$ .

$\longrightarrow y_1 = x_1$  vrai positif,  $y_2 (\neq x_1)$  faux positif

# Fonctions de hachage

Fonction  $h : U \longrightarrow [1 \dots M]$  avec  $M < |U|$ .

La fonction ventile "aléatoirement" les éléments de  $U$ .

## Remplissage

On représente  $S = \{x_1, \dots, x_n\}$  par  $S' = \{h(x_j) \mid x_j \in S\}$

→ mise à 1 des positions  $h(x_j)$  dans la mémoire

# Fonctions de hachage

Fonction  $h : U \longrightarrow [1 \dots M]$  avec  $M < |U|$ .

La fonction ventile "aléatoirement" les éléments de  $U$ .

## Remplissage

On représente  $S = \{x_1, \dots, x_n\}$  par  $S' = \{h(x_j) \mid x_j \in S\}$   
→ mise à 1 des positions  $h(x_j)$  dans la mémoire

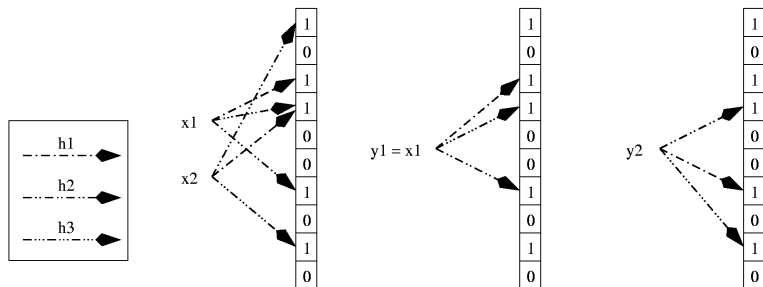
## Utilisation

On prédit que  $y \in S$  si  $h(y) \in S'$   
→ interrogation de la mémoire à la position  $h(y)$

Taux de faux positifs :  $(1 - (1 - 1/M)^n) \sim 1 - e^{-n/M}$

# Filtres Bloom

Bloom, puis Carter et al. (1970's)



3 fonctions de hachage,  $S = \{x_1, x_2\}$ ,  $M = 10$ .

→  $y_1 = x_1$  vrai positif,  $y_2 (\neq x_1)$  faux positif

$d$  fonctions de hachage différentes  $h_1, h_2, \dots, h_d$ .

## Remplissage

On représente  $S = \{x_1, \dots, x_n\}$  par  $S' = \{h_i(x_j) \mid x_j \in S\}$   
→ mise à 1 des positions  $h_i(x_j)$  dans la mémoire

# Filtres Bloom

$d$  fonctions de hachage différentes  $h_1, h_2, \dots, h_d$ .

## Remplissage

On représente  $S = \{x_1, \dots, x_n\}$  par  $S' = \{h_i(x_j) \mid x_j \in S\}$   
→ mise à 1 des positions  $h_i(x_j)$  dans la mémoire

## Utilisation

On prédit que  $y \in S$  si  $\forall i, h_i(y) \in S'$   
→ interrogation de la mémoire aux positions  $h_i(y)$

Taux de faux positifs :  $(1 - (1 - 1/M)^{dn})^d \sim (1 - e^{-dn/M})^d$

# Avantages des filtres Bloom

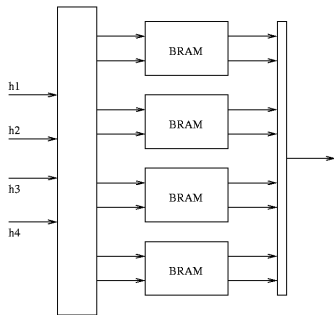
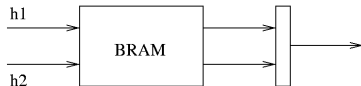
- Le taux de faux positifs est inférieur à celui d'une simple fonction de hachage...
- ... il est donc possible de stocker plus de données avec la même mémoire et le même taux d'erreur.
- La mémoire nécessaire est de  $n \log_2(1/\epsilon) \log_2 e$ , soit seulement un facteur de  $\log_2 e$  au-dessus de l'optimum théorique.
- Meilleur choix du nombre de fonctions de hachage :  
 $d = \ln 2 \cdot (M/n)$ .

## Mémoire

12 BRAM de 512 octets ( $= 2^{12} = 4096$  bits) chacun  
8 BRAM disponible pour le filtre.

- duplication de mémoire
- accès dual-port [Dharma, 2004], accès partagé

# Accès multiples aux BRAM



2 à 4 accès mémoire par cycle

## Mémoire

12 BRAM de 512 octets ( $= 2^{12} = 4096$  bits) chacun  
8 BRAM disponible pour le filtre.

- duplication de mémoire
- accès dual-port [Dharma, 2004], accès partagé

## Fonctions de hachage

À base de XOR [Rama 94].

$$h_q(x) = x_1 \cdot q(1) \oplus x_2 \cdot q(2) \oplus x_3 \cdot q(3) \oplus \dots \oplus x_n \cdot q(n)$$

→ une position par cycle (@ 40 MHz) → 40 Mbp/s

# Utilisation des filtres Bloom

- Graines de poids  $w$  ( $2w$  bits) :  $|U| = 2^{2w}$  bits
- Mémoire totale sur FPGA
  - $2^{15}$  bits accessible à chaque cycle
  - insuffisant pour des graines de taille  $\geq 8$

# Utilisation des filtres Bloom

- Graines de poids  $w$  ( $2w$  bits) :  $|U| = 2^{2w}$  bits
- Mémoire totale sur FPGA
  - $2^{15}$  bits accessible à chaque cycle
  - insuffisant pour des graines de taille  $\geq 8$
- hachage simple → 1/50 faux positifs

# Utilisation des filtres Bloom

- Graines de poids  $w$  ( $2w$  bits) :  $|U| = 2^{2w}$  bits
- Mémoire totale sur FPGA
  - $2^{15}$  bits accessible à chaque cycle
  - insuffisant pour des graines de taille  $\geq 8$
- hachage simple → 1/50 faux positifs
- 4 fonctions de hachage, mémoire repliquée de  $2^{14}$  bits avec 2 ou 4 accès multiples
  - moins de 1/400 faux positifs
- La taille des graines n'est impliquée que dans les fonctions de hachage !
- graines de taille 8 – 16
  - le hachage occupe  $< 10\%$  du FPGA
  - place pour un étage d'extension

# Est-ce qu'un ordinateur fait encore des calculs ?

## Recherche de similarités entre $m$ et $n$

pour chaque graine  $s \in m$

**si**  $s$  apparaît dans  $n$

**alors**

soit  $v_m$  et  $v_n$  les voisinages de  $s$  dans  $m$  et  $n$

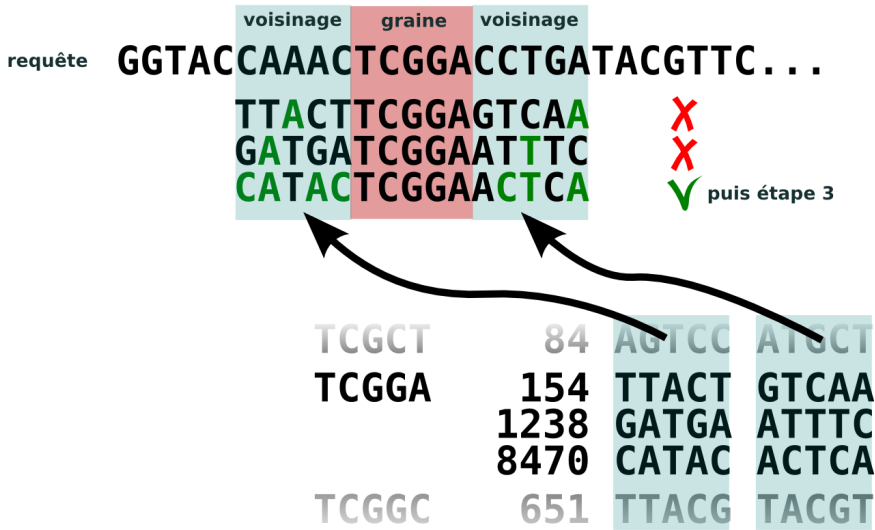
**si**  $v_m$  et  $v_n$  s'alignent sommairement

**alors**

soit  $V_m$  et  $V_n$  des voisinages plus étendus  
aligner  $V_m$  et  $V_n$

*parcours aléatoire de la séquence  $n$  ?*

# Indexation des graines, mémorisation des voisinages



- Algorithmes, mémoire et architecture
  - Localité mémoire
  - Algorithmes efficaces pour la mémoire...
  - ... ou du moins conscients de leurs limites
  - Circuits spécialisés, FPGA
- Architectures systoliques
  - Programmation dynamique
  - Fonctionnement systolique
  - Optimisations
- Heuristiques à base de graines
  - Heuristiques
  - Représentations d'ensembles, filtres Bloom
  - Amélioration des graines

# Indexation des graines, mémorisation des voisinages

voisinage graine voisinage

requête **GGTACCAA**CTCGGACCTGATACGTTC...

154 ... **TTACTTCGGAGTCAA**... X

1238 ... **GATGATCGGAATTTC**... X

8470 ... **CATAC**TCGGAACTCA... ✓ puis étape 3

XXXXX#####XXXXX

TCGCT	84	AGTCC	ATGCT
TCGGA	154	TTACT	GTCAA
	1238	GATGA	ATTTC
	8470	CATAC	ACTCA
TCGGC	651	TTACG	TACGT

# Indexation des graines, mémorisation des voisinages

voisinage graine voisinage

requête **GGTACCAA**CTCGGAC**CTGATACGTTC...**

**761** .. **CTAA**ACTGAGT**ACTTA**... ✓ puis étape 3

**5109** .. **AATTT**CTCTGAGG**GAGA**... ✗

**8470** .. **CCATA**CTCGGA**ACTCA**... ✓ puis étape 3

XXXXX###-##XXXXX

	84	AGTCC	ATGCT
<b>CTC-GA</b>	<b>761</b>	<b>CTAAA</b>	<b>ACTTA</b>
	<b>5109</b>	<b>AATTT</b>	<b>CGAGA</b>
	<b>8470</b>	<b>CCATA</b>	<b>ACTCA</b>
<b>CTC-GC</b>	<b>4771</b>	<b>TTACG</b>	<b>TACGT</b>

# Indexation des graines, mémorisation des voisinages

requête **GGTACCAA**CTC**GGACCTGATACGTTC...**

	voisinage	graine	voisinage	
761	CTAAACTG	GTACTTA	...	✓ puis étape 3
5109	AATTTCTC	GAGGAGA	...	✗
8470	CCATACTC	GAACTCA	...	✓ puis étape 3

XXXXX###-##XXXXX

	84	AGTCC	ATGCT
CTC-GA	761	CTAAA	ACTTA
	5109	AATTT	CGAGA
	8470	CCATA	ACTCA
CTC-GC	4771	TTACG	TACGT

# Graines contiguës ou espacées

ATCAGTGCAATGCTCAAGA

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

ATCAGCGCGATGCGCAAGA

#####

ATCAGTGCAATGCTCAAGA

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

ATCAGCGCGATGCGCAAGA

###--#-##

# Graines contiguës ou espacées

ATCAGTGCAATGCTCAAGA

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

ATCAGCGCGATGCGCAAGA

#####

#####

ATCAGTGCAATGCTCAAGA

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

ATCAGCGCGATGCGCAAGA

###--#-##

###--#-##

# Graines contiguës ou espacées

ATCAGTGCAATGCTCAAGA

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

ATCAGCGCGATGCGCAAGA

#####

#####

#####

ATCAGTGCAATGCTCAAGA

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

ATCAGCGCGATGCGCAAGA

###--#-##

###--#-##

###--#-##

# Graines contiguës ou espacées

ATCAGTGCAATGCTCAAGA

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

ATCAGCGCGATGCGCAAGA

#####

#####

#####

#####

#####

#####

#####

#####

#####

#####

#####

#####

#####

#####

ATCAGTGCAATGCTCAAGA

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

ATCAGCGCGATGCGCAAGA

###-#-##

###-#-##

###-#-##

###-#-##

###-#-##

###-#-##

###-#-##

###-#-##

###-#-##

###-#-##

###-#-##

# Graines contiguës ou espacées

ATCAGTGCAATGCTCAAGA

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

ATCAGCGCGATGCGCAAGA

###--#-##

###--#-##

###--#-##

Les graines espacées sont plus sensibles que les graines contiguës.

# Graines contiguës ou espacées

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

###--#-##

###--#-##

###--#-##

Les graines espacées sont plus sensibles que les graines contiguës.

# Amélioration des graines

Plus le  $w$  est petit, plus on détecte d'alignements, mais plus l'étape 2 est surchargée.

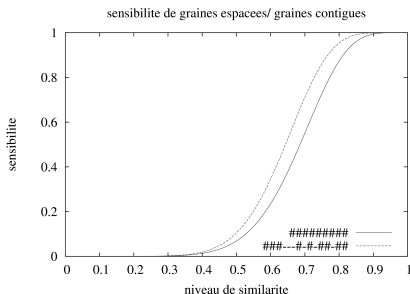
- Graines contigües : ATATTTACGTG ( $w = 11$ )

# Amélioration des graines

Plus le  $w$  est petit, plus on détecte d'alignements, mais plus l'étape 2 est surchargée.

- Graines contigües : ATATTTACGTG ( $w = 11$ )
- Graines espacées : A-TTGC-TT-ATTG  
(#-####-##-####, poids 11, étendue 14)

- Graines vecteurs
- Graines sous-ensemble
- Graines multiples



# Conception de graines espacées

Conception de graines sur l'alphabet  $\{ -, \# \}$

- aléatoire (Flash, Buhler)

# Conception de graines espacées

Conception de graines sur l'alphabet { -, # }

- aléatoire (Flash, Buhler)
- détectant le plus d'alignements (PatternHunter)

###---#-#-##-##

# Conception de graines espacées

Conception de graines sur l'alphabet  $\{ -, \# \}$

- aléatoire (Flash, Buhler)
- détectant le plus d'alignements (PatternHunter)

###---#-#-##-##

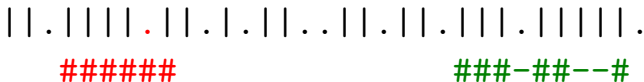
- détectant tous les alignements  
avec un nombre maximum de substitutions (Burkhardt et  
Kärkkäinen)

###-#--###-#--###-#

(poids 12, tous les alignements de taille  $\geq 25$  avec  $\leq 2$  substitutions)

# Alphabets d'alignements et de graines

	mismatch	match
	30%	70%
	.	
-	o	o
#		o







Comment réaliser en matériel...

- ... des graines espacées ?
- ... des graines avec un décompte des positions ?
- ... des graines multiples ?

# Machines proposées

- BLAST, mpiBLAST...
- BioSCAN (1993) (étape 1 ?)

- BLAST, mpiBLAST...
- BioSCAN (1993) (étape 1 ?)
- FPGA : BEE2 (2000), Mercury (2002), RC-BLAST (2005)
- autres heuristiques : IRISA / Rdisk (2003), DASH (2004)

→ Il y a très peu d'architectures accélérant d'autres calculs bioinformatique que la comparaison de séquences.

- Algorithmes, mémoire et architecture
  - Localité mémoire
  - Algorithmes efficaces pour la mémoire...
  - ... ou du moins conscients de leurs limites
  - Circuits spécialisés, FPGA
- Architectures systoliques
  - Programmation dynamique
  - Fonctionnement systolique
  - Optimisations
- Heuristiques à base de graines
  - Heuristiques
  - Représentations d'ensembles, filtres Bloom
  - Amélioration des graines

## Architectures pour la génomique

- Programmation dynamique : architectures systoliques  
ASIC (1985), FPGA (1990, 2000 –)  
→ OK, encodage modulo : réducteur?
- Heuristiques à base de graines  
compromis sensibilité / vitesse de calcul  
→ souvent réduit à Blast
- Il n'y a pas que la comparaison de séquences!  
→ modèles, motifs, automates, grammaires...

## Architectures pour la génomique

- Programmation dynamique : architectures systoliques  
ASIC (1985), FPGA (1990, 2000 –)  
→ OK, encodage modulo : réducteur ?
  - Heuristiques à base de graines  
compromis sensibilité / vitesse de calcul  
→ souvent réduit à Blast
  - Il n'y a pas que la comparaison de séquences !  
→ modèles, motifs, automates, grammaires...
- 
- Architecture : complexités en espace, chemin critique, synchronisation des données
  - Besoin de solutions théoriques aux problèmes en architecture
  - Temps de conception et de déploiement ?