Alignement rapide avec hachage

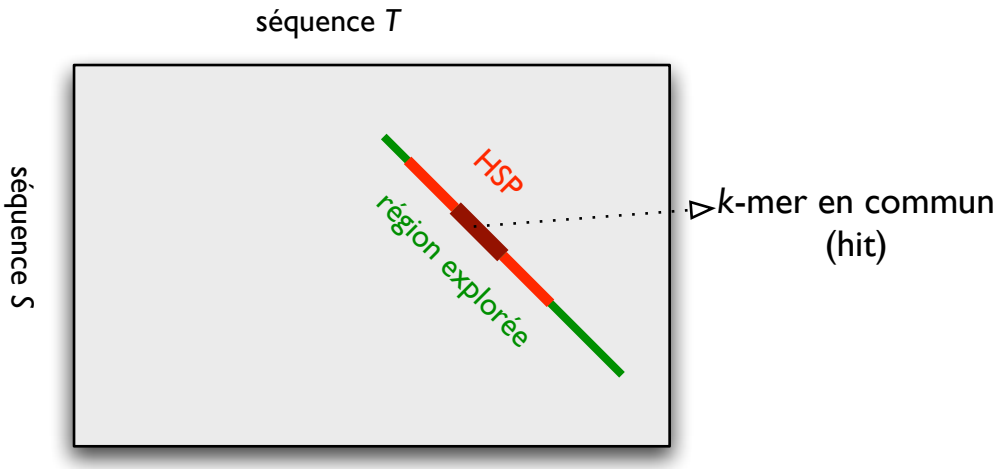
Recherche de similarités

Étant donné un génome S et un autre génome ou morceau séquencé T, on veut trouver des similarités $S[i..i+\delta] \sim T[j..j+\delta]$

Pour n'importe quel i, j, δ, on peut calculer le score (basé sur LODS comme avant)

1. comment trouver (i, j) ? [indexage/hachage]
2. étant donné i et j, comment choisir δ ? [extension]
3. évaluation statistique de cette méthodologie [P-value ?]

Extension rapide



Rester sur la même diagonale ; explorer jusqu'à ce que le score tombe sous un seuil t, prendre le meilleur segment (*high-scoring segment pair*, HSP)

X-drop : explorer jusqu'à ce que le score tombe par un seuil X

Hachage

Deux séquences S, T

Fonction de hachage : $h : \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}^k \mapsto \mathcal{H}$

hit : (i, j) avec $h(S[i..i+k-1]) = h(T[j..j+k-1])$

Technique : listes $\mathsf{Occ}(u)$ de positions où $h^{-1}(u)$ apparaît

1. **pour** $i \leftarrow 1, \ldots, |S| - k + 1$ **faire**
2. $\quad \mathsf{clé} \leftarrow h(S[i..i+k-1])$
3. $\quad$ ajouter i à la fin de la liste $\mathsf{Occ}(\mathsf{clé})$
4. **pour** $j \leftarrow 1, \ldots, |T| - k + 1$ **faire**
5. $\quad \mathsf{clé} \leftarrow h(T[j..j+k-1])$
6. $\quad$ traitement [extension ?] des *hits* $(i, j) : i \in \mathsf{Occ}(\mathsf{clé})$

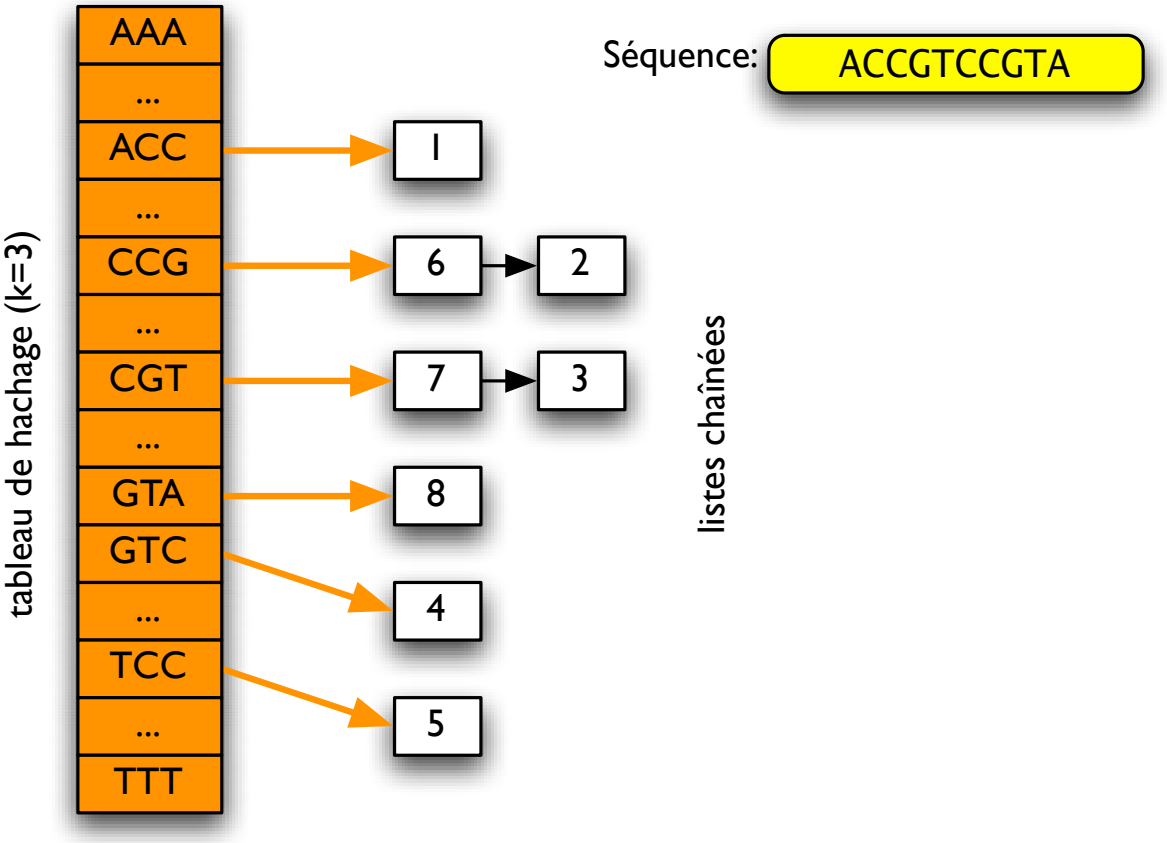
Hachage — structure de données

Trouver les clés partagés : stocker les occurrences (Occ) de tous les clés de S en un tableau de hachage

Implantation facile en Java : on peut utiliser les sous-mots w comme clés directement, `Hashtable` calcule des clés de hachage automatiquement, liste chaînée pour chaque $\mathsf{Occ}(w)$

(utilise plus de mémoire que nécessaire...)

Tableau de k-mers



Ma, Tromp et Li. *Bioinformatics* 18 :440.

Implantation

1. Encodage des k-mers en $2k$ bits :
$\texttt{A} \rightarrow \texttt{00}, \texttt{C} \rightarrow \texttt{01}, \texttt{G} \rightarrow \texttt{10}, \texttt{T} \rightarrow \texttt{11}$.

Java `int` : 32 bits ($k \leq 16$) ; `long` : 64 bits ($k \leq 32$).

2. Encodage des listes chaînées :
chaque position de la séquence n'apparaît qu'une fois !
Définir un tableau `int[] successeur` où `successeur`$[i]$ donne la position qui suit i dans une des listes chaînées ou égale à -1 si i est la dernier objet dans une liste.
Tête de chaque liste est trouvée par un tableau `int[] tete` où `tete`$[i]$ donne la première position ou le k-mer encodé par i se trouve.

Mémoire : $(4^k + |S|)$ fois taille de `int` (4 octets).

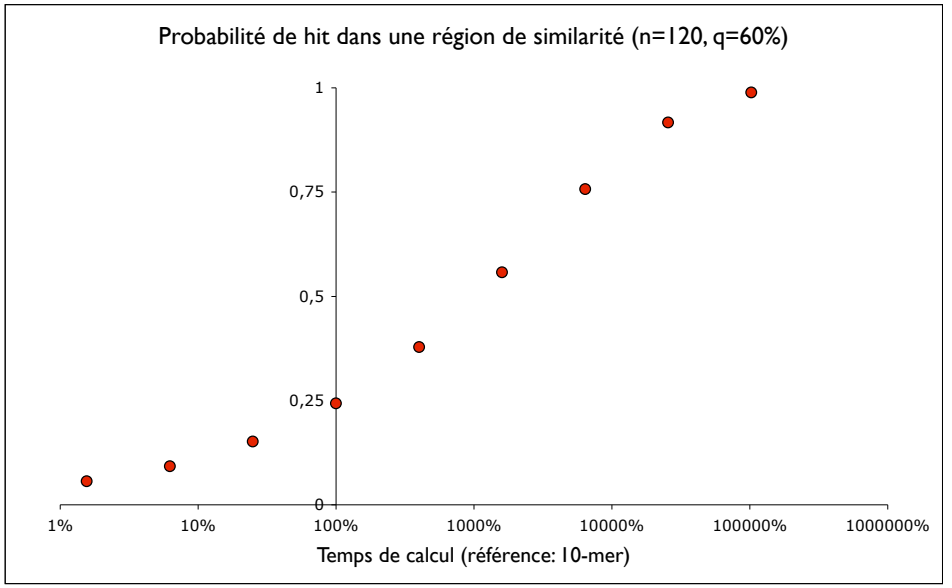
Implantation — Java

```
HT1  int[] tete=new int[1<<(2*k)];
HT2  int[] successeur=new int[S.length()-k+1];
HT3  pour tout i, tete[i] ← −1
HT4  for (int i=0; i<S.length()-k+1; i++){
HT5      calcul de l'encodage w pour le sous-mot S[i..i + k − 1];
HT6      successeur[i]=tete[w];
HT7      tete[w]=i;
HT8  }
HT9  for (int j=0; j<T.length()-k+1; j++){
HT10     calcul de l'encodage w pour le sous-mot T[j..j + k − 1];
HT11     int i=tete[w];
HT12     while (i != -1){
HT13         extension du hit (i, j)
HT14         i=successeur[i];
HT15     }
HT16 }
```

Hachage — performance

Spécificité : mesurée par nombre de *hits* entre deux séquences sans homologies (p.e., aléatoires)

Sensitivité : mesurée par la probabilité de *hit* dans une région de homologie



Hachage — nombre de *hits*

Modèle : S aléatoire, avec nucléotides iid selon p ; T aléatoire, avec nucléotides iid selon q : $\mathbb{P}\{S[i] = c\} = p_c$ et $\mathbb{P}\{T[j] = c'\} = q_{c'}$.

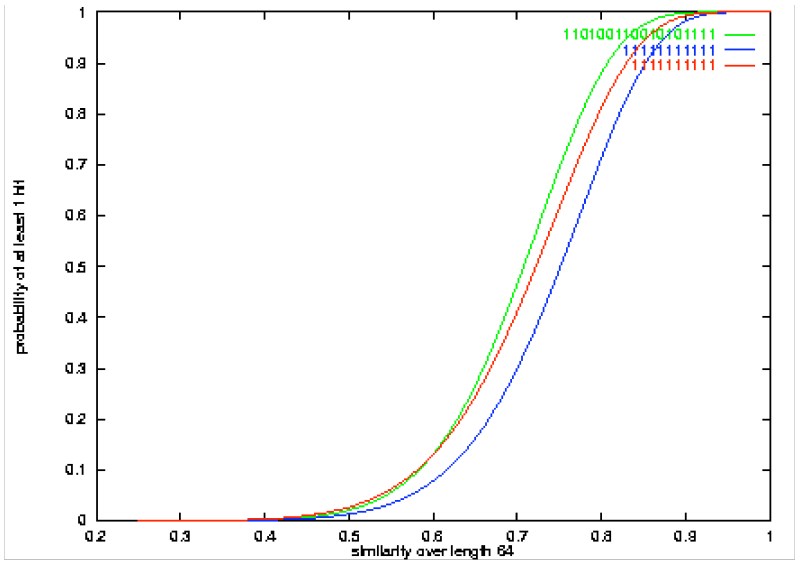
Fonction de hachage : identité $h(u) = u$, $\mathcal{H} = \Sigma^k$ ($\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$)

Thm. Soit $\beta = \sum_{c \in \Sigma} p_c q_c$. Alors le nombre de *hits* en espérance est $st\beta^k$ où $s = |S| - k + 1$ et $t = |T| - k + 1$.

Meilleur hachage : graines espacées

$\mathcal{S} = \{s_1, \ldots, s_k\}$: graine (*seed*) (ℓ, k) ;
fonction de hachage : $h(u) = u[s_1] \cdot u[s_2] \cdots u[s_k]$.

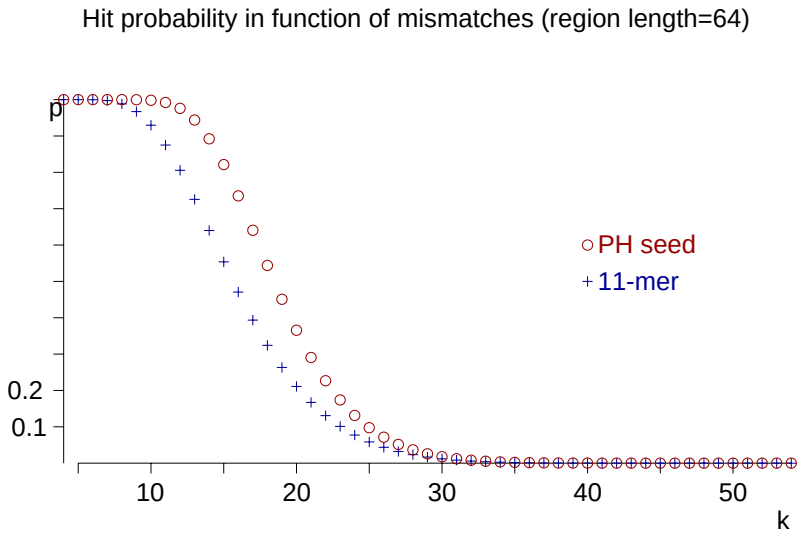
BLAST : graine contigue (11-mer) $\mathcal{S} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$
PatternHunter : graine espacée $\mathcal{S} = \{1, 2, 3, 5, 8, 10, 13, 14, 16, 17, 18\}$



Ma et al. *Bioinformatics* 18 :440

Graines espacées

Graine espacée est meilleure pour toutes distributions de similarité



Problème : trouver la meilleure graine (NP-difficile), calculer la sensitivité (NP-difficile)

Calculer la sensitivité de la graine

Modèle : région de similarité de longueur ℓ, niveau de similarité p. Séquence d'indicateurs $X_1 X_2 \cdots X_\ell$ où $X_i = 1$ ssi un match en position i (avec probabilité p).

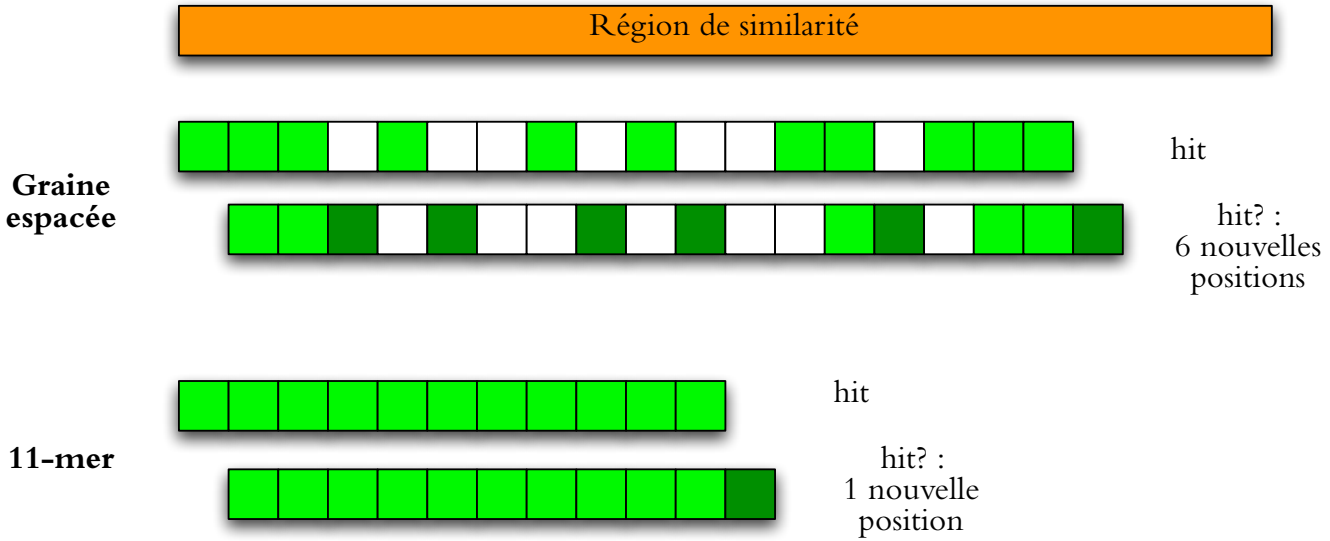
Hit en position i : si $X_{i+k-1} = 1$ pour tout $k \in \mathcal{S}$.

Calculer cette probabilité ...

Indépendance de positions

Nombre de hits en espérance à peu près la même

hits sont plus dense avec graine contigue

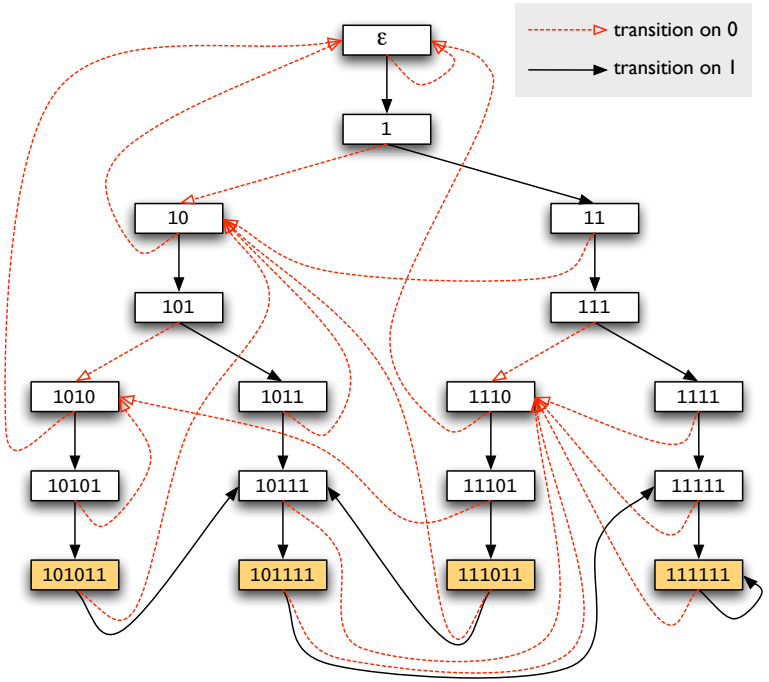


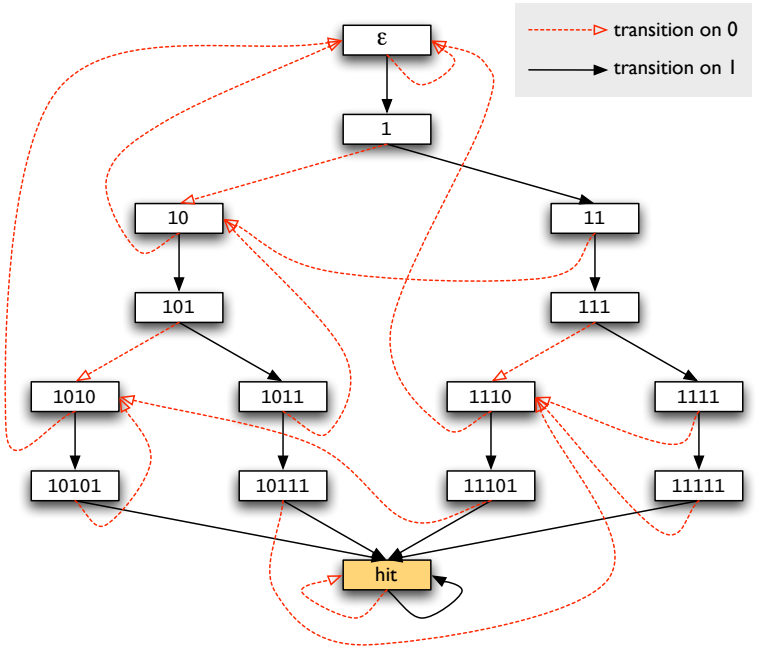
probabilité d'un deuxième hit : p^6 ou p^1

nombre de hits H : $\mathbb{E}H = \mathbb{E}\left[H \;\middle|\; H > 0\right]\mathbb{P}\{H > 0\}$

or, $\mathbb{E}\left[H \;\middle|\; H > 0\right]$ est plus large avec k-mer

Sensitivité de la graine 2





Graine = `101011`