Quelques slides sur les gr__n_s

dans la série : "hangman game for dummies"

Laurent

LIFL, Université Lille 1 - INRIA

Journées au vert

20 et 21 mai 2011

NGS : encore de nouvelles versions en “scale down”

mais je n’en parlerai pas . . .

source : `http://www.illumina.com/systems/miseq.ilmn`

Alors, de quoi vais-je bien pouvoir parler ?

Une idée ??

Graines espacées

(*PatternHunter 02, Burkhardt et al 01, BLASTz 03, YASS 04*)

Definition

Une graine espacée π est un mot binaire sur l'alphabet $\{\#, -\}$:

- `#` : accepte seulement un symbole *match* (`|`).
- `-` : accepte un symbole *match* (`|`) ou un symbole *mismatch* (`.`).

s : *span* (longueur), w : *weight* (nombre de `#`).

Example

graine $\pi =$ `###-#-##`

```
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.||||
ATCAGCGCAAATGCTCAAGA
```

Graines espacées

(*PatternHunter 02, Burkhardt et al 01, BLASTz 03, YASS 04*)

Definition

Une graine espacée π est un mot binaire sur l'alphabet $\{\#, -\}$:

- `#` : accepte seulement un symbole *match* (`|`).
- `-` : accepte un symbole *match* (`|`) ou un symbole *mismatch* (`.`).

s : *span* (longueur), w : *weight* (nombre de `#`).

Example

graine $\pi =$ `###-#-##`

```
      ###-#-##
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
```

Graines espacées

(*PatternHunter 02, Burkhardt et al 01, BLASTz 03, YASS 04*)

Definition

Une graine espacée π est un mot binaire sur l'alphabet $\{\#, -\}$:

- `#` : accepte seulement un symbole *match* (`|`).
- `-` : accepte un symbole *match* (`|`) ou un symbole *mismatch* (`.`).

s : *span* (longueur), w : *weight* (nombre de `#`).

Example

graine $\pi =$ `###-#-##`

```
      ###-#-##
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
```

Graines espacées

(*PatternHunter 02, Burkhardt et al 01, BLASTz 03, YASS 04*)

Definition

Une graine espacée π est un mot binaire sur l'alphabet $\{\#, -\}$:

- `#` : accepte seulement un symbole *match* (`|`).
- `-` : accepte un symbole *match* (`|`) ou un symbole *mismatch* (`.`).

s : *span* (longueur), w : *weight* (nombre de `#`).

Example

graine $\pi =$ `###-#-##`

```
      ###-#-##
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
```

Graines espacées

(*PatternHunter 02, Burkhardt et al 01, BLASTz 03, YASS 04*)

Definition

Une graine espacée π est un mot binaire sur l'alphabet $\{\#, -\}$:

- `#` : accepte seulement un symbole *match* (`|`).
- `-` : accepte un symbole *match* (`|`) ou un symbole *mismatch* (`.`).

s : *span* (longueur), w : *weight* (nombre de `#`).

Example

graine $\pi =$ `###-#-##`

```
###-#-##
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
```

Graines espacées

(*PatternHunter 02, Burkhardt et al 01, BLASTz 03, YASS 04*)

Definition

Une graine espacée π est un mot binaire sur l'alphabet $\{\#, -\}$:

- `#` : accepte seulement un symbole *match* (`|`).
- `-` : accepte un symbole *match* (`|`) ou un symbole *mismatch* (`.`).

s : *span* (longueur), w : *weight* (nombre de `#`).

Example

graine $\pi =$ `###-#-##`

```
###-#-##
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
```

Graines espacées

(*PatternHunter 02, Burkhardt et al 01, BLASTz 03, YASS 04*)

Definition

Une graine espacée π est un mot binaire sur l'alphabet $\{\#, -\}$:

- `#` : accepte seulement un symbole *match* (`|`).
- `-` : accepte un symbole *match* (`|`) ou un symbole *mismatch* (`.`).

s : *span* (longueur), w : *weight* (nombre de `#`).

Example

graine $\pi =$ `###-#-##`

```
###-#-##
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
```

Example

Example

$$\pi_c = \texttt{######}$$

$$\pi_s = \texttt{###--\#-\#\#}$$

Example

$$\pi_c = \texttt{######}$$

$$\pi_s = \texttt{###--\#-\#\#}$$

$$\alpha = \begin{array}{l} \texttt{ATCAGTGCAAATGCTCAAGA} \\ \texttt{||||||||||||||||||||} \\ \texttt{ATCAGTGCAAATGCTCAAGA} \end{array}$$

Example

$$\pi_c = \texttt{\#\#\#\#\#\#}$$

$$\pi_s = \texttt{\#\#\#--\#-\#\#}$$

$$\alpha = \begin{array}{l} \texttt{ATCAGTGCAAATGCTCAAGA} \\ \texttt{||||||||||||||||||||} \\ \texttt{ATCAGTGCAAATGCTCAAGA} \end{array}$$

Example

$$\pi_c = \texttt{\#\#\#\#\#\#}$$

$$\pi_s = \texttt{\#\#\#--\#-\#\#}$$

$$\alpha = \begin{array}{l} \texttt{ATCAGTGCAAATGCTCAAGA} \\ \texttt{||||| ||||||||||||||} \\ \texttt{ATCAGCGCAAATGCTCAAGA} \end{array}$$

Example

$$\pi_c = \texttt{\#\#\#\#\#\#}$$

$$\pi_s = \texttt{\#\#\#--\#-\#\#}$$

$$\alpha = \begin{array}{l} \texttt{ATCAGTGCAAATGCTCAAGA} \\ \texttt{||||| |||||||||||||} \\ \texttt{ATCAGCGCAAATGCTCAAGA} \end{array}$$

Example

$$\pi_c = \texttt{\#\#\#\#\#\#}$$

$$\pi_s = \texttt{\#\#\#--\#-\#\#}$$

$$\alpha = \begin{array}{l} \texttt{ATCAGTGCAAATGCGCAAGA} \\ \texttt{||||| |||||||| |||||} \\ \texttt{ATCAGCGCAAATGCTCAAGA} \end{array}$$

Example

$$\pi_c = \texttt{######}$$

$$\pi_s = \texttt{###--\#-\#\#}$$

$$\alpha = \begin{array}{l} \texttt{ATCAGTGCAAATGCGCAAGA} \\ \texttt{||||| ||||||| |||||} \\ \texttt{ATCAGCGCAAATGCTCAAGA} \end{array}$$

Example

$$\pi_c = \texttt{######}$$

$$\pi_s = \texttt{###--#-##}$$

$$\alpha = \begin{array}{l} \texttt{ATCAGTGCGAATGCGCAAGA} \\ \texttt{||||| || ||||| |||||} \\ \texttt{ATCAGCGCAAATGCTCAAGA} \end{array}$$

Example

Example

```
ATCAGTGCAAATGCTCAAGA
||||||||||||||||||||
ATCAGTGCAAATGCTCAAGA
```

```
ATCAGTGCAAATGCTCAAGA
||||||||||||||||||||
ATCAGTGCAAATGCTCAAGA
```

Example

```
ATCAGTGCAAATGCTCAAGA
||||||||||||||||||||
ATCAGTGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
```

```
ATCAGTGCAAATGCTCAAGA
||||||||||||||||||||
ATCAGTGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
```

Example

```
ATCAGTGCAAATGCTCAAGA
|||||||||||||||||||||
ATCAGTGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
               ######
```

```
ATCAGTGCAAATGCTCAAGA
|||||||||||||||||||||
ATCAGTGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
```

Example

```
ATCAGTGCAAATGCTCAAGA
|||||.||||||||||||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
```

```
ATCAGTGCAAATGCTCAAGA
|||||.||||||||||||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
```

Example

```
ATCAGTGCAAATGCTCAAGA
|||||.||||||||||||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
```

```
ATCAGTGCAAATGCTCAAGA
|||||.||||||||||||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
```

Example

```
ATCAGTGCAAATGCTCAAGA
|||||.||||||||||||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
```

```
ATCAGTGCAAATGCTCAAGA
|||||.||||||||||||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
```

Example

```
ATCAGTGCAAATGCTCAAGA
|||||.||||||||||||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
```

```
ATCAGTGCAAATGCTCAAGA
|||||.||||||||||||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
```

Example

```
ATCAGTGCAAATGCGCAAGA
|||||.|||||||||.|||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
```

```
ATCAGTGCAAATGCGCAAGA
|||||.|||||||||.|||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
```

Example

```
ATCAGTGCAAATGCGCAAGA
|||||.|||||||||.|||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          #####
           ######
            ######
             ######
              ######
               ######
```

```
ATCAGTGCAAATGCGCAAGA
|||||.|||||||||.|||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
```

Example

```
ATCAGTGCAAATGCGCAAGA
|||||.|||||||||.|||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
               ######
```

```
ATCAGTGCAAATGCGCAAGA
|||||.|||||||||.|||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ##--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
```

Example

```
ATCAGTGCAAATGCGCAAGA
|||||.|||||||.|||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
               ######
```

```
ATCAGTGCAAATGCGCAAGA
|||||.|||||||.|||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
             ###--#-##
              ###--#-##
               ###--#-##
```

Example

```
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
               ######
```

```
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
             ###--#-##
              ###--#-##
               ###--#-##
```

Example

```
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       #####
        #####
         ######
          ######
           ######
            ######
             ######
              ######
               ######
```

```
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
```

Example

```
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
               ######
```

```
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
             ###--#-##
              ###--#-##
               ###--#-##
```

Example

```
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
######
 ######
  ######
   ######
    ######
     ######
      ######
       ######
        ######
         ######
          ######
           ######
            ######
             ######
              ######
               ######
```

```
ATCAGTGCGAATGCGCAAGA
|||||.||.||||||.|||||
ATCAGCGCAAATGCTCAAGA
###--#-##
 ###--#-##
  ###--#-##
   ###--#-##
    ###--#-##
     ###--#-##
      ###--#-##
       ###--#-##
        ###--#-##
         ###--#-##
          ###--#-##
           ###--#-##
            ###--#-##
```

Graines espacées

langage d'une graine

Besoin de connaître le *language* reconnu par la graine π.

Graines espacées

langage d'une graine

Besoin de connaître le *language* reconnu par la graine π.

Alignement : mot sur l'alphabet binaire *match* / *mismatch*

Graines espacées

langage d'une graine

Besoin de connaître le *language* reconnu par la graine π.

Alignement : mot sur l'alphabet binaire *match* / *mismatch*

Notation : $\{1, 0\}$

Calculer la sensibilité d'une graine

exemple utilisant l'automate Aho-Corasick (Buhler et al 2003)

graine $\pi =$ #-#-#

Calculer la sensibilité d'une graine

exemple utilisant l'automate Aho-Corasick (Buhler et al 2003)

graine $\pi =$ #-#-#

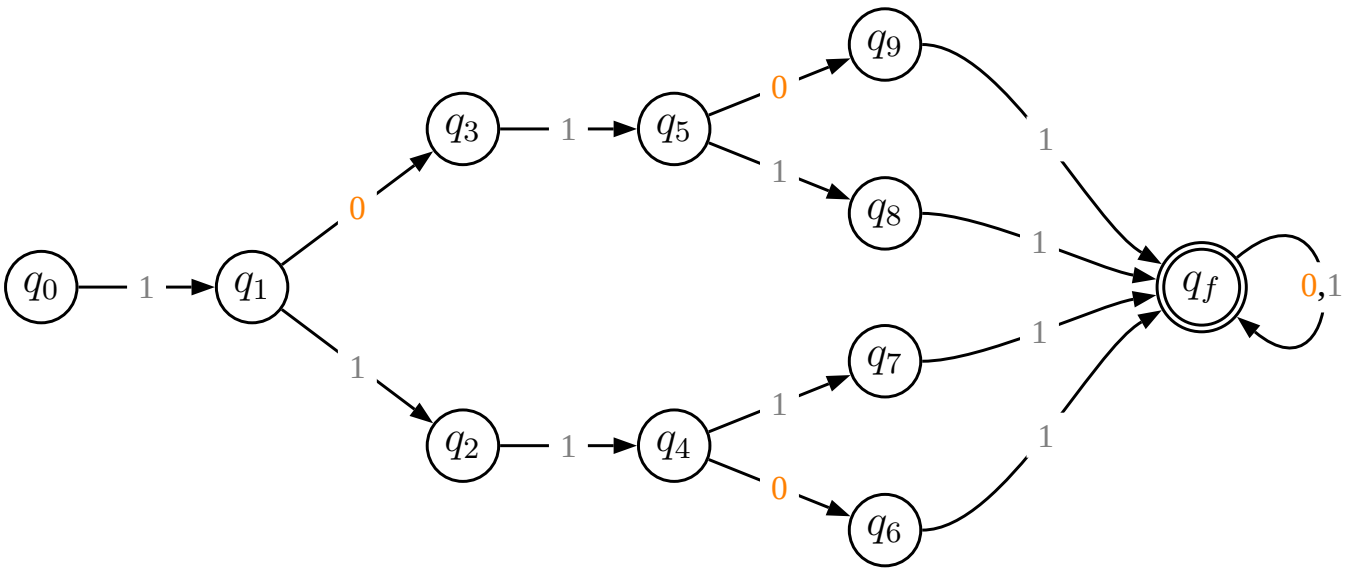
fragments d'alignement detectés : {10101, 10111, 11101, 11111}

Calculer la sensibilité d'une graine

exemple utilisant l'automate Aho-Corasick (Buhler et al 2003)

graine $\pi =$ #-#-#

fragments d'alignement detectés : {10101, 10111, 11101, 11111}

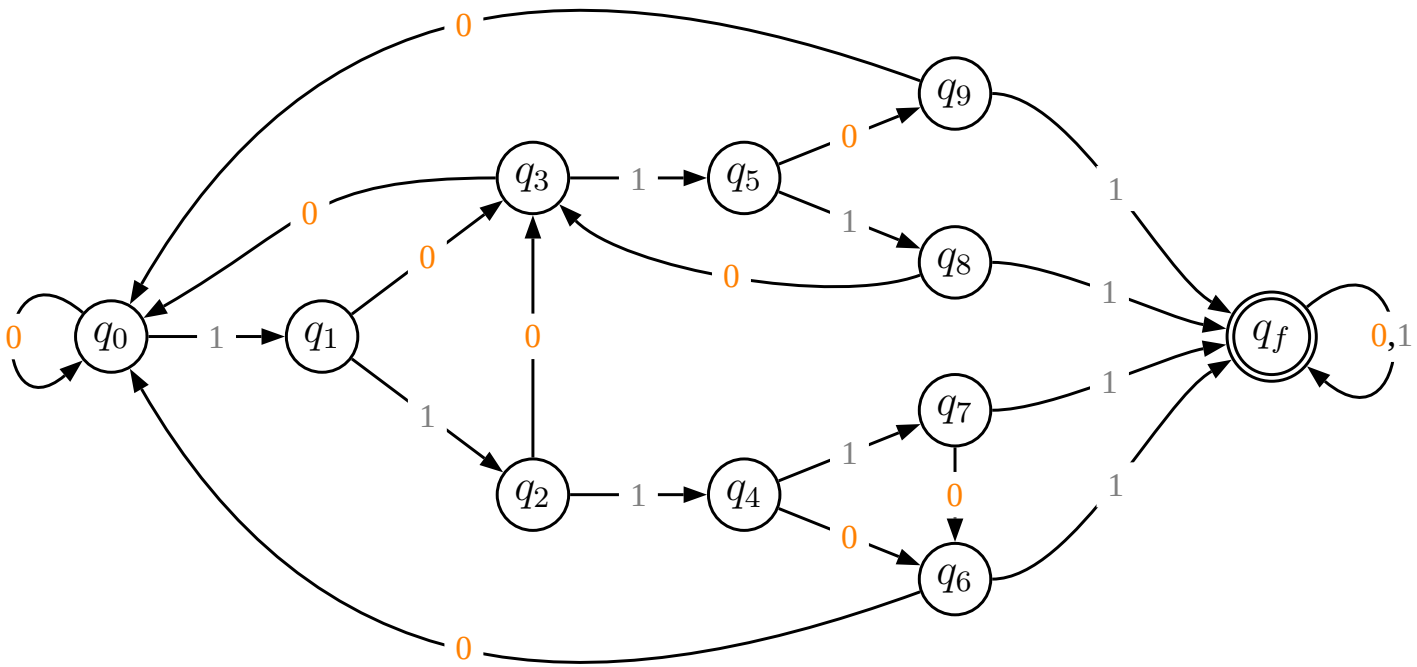


Calculer la sensibilité d'une graine

exemple utilisant l'automate Aho-Corasick (Buhler et al 2003)

graine π = #-#-#

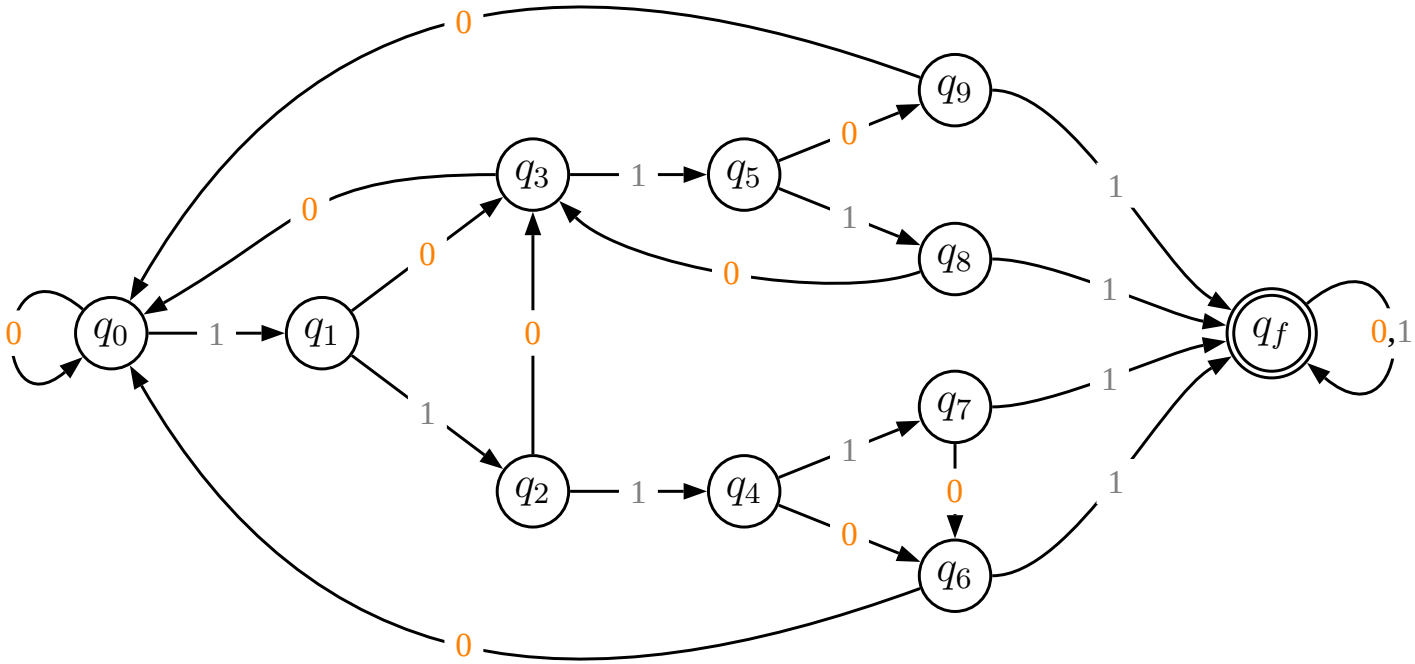
fragments d'alignement detectés : {10101, 10111, 11101, 11111}



Calculer la sensibilité d'une graine

exemple utilisant l'automate Aho-Corasick (Buhler et al 2003)

- en Lossy (exemple du modèle de Bernoulli sur un alignment de taille m)
  semi-anneau $(E = \mathbb{R}_{\geq 0}, \oplus = +, \otimes = \times, 0_{\oplus} = 0, 1_{\otimes} = 1)$
- en Lossless (exemple du modèle "k erreurs" sur un alignment de taille m)
  semi-anneau *tropical* $(E = \mathbb{N}, \oplus = min, \otimes = +, 0_{\oplus} = \infty, 1_{\otimes} = 0)$



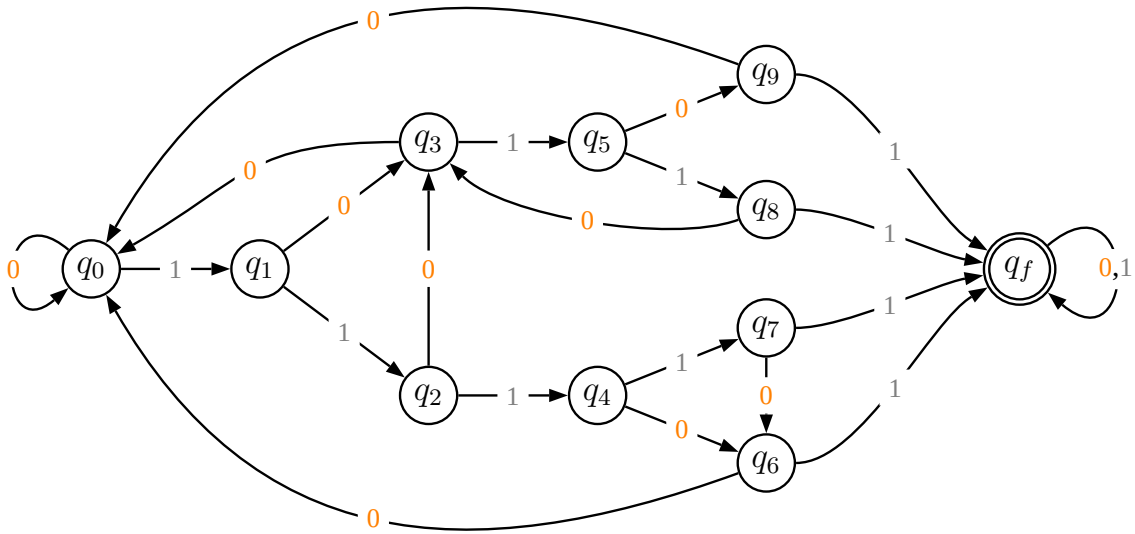
Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses

$M =$

	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_f
q_0	0	1									
q_1			0	1							
q_2				0	1						
q_3	0					1					
q_4							0	1			
q_5									1	0	
q_6	0										1
q_7							0				1
q_8				0							1
q_9	0										1
q_f											0,1

$A =$



Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses et produit

- Sensibilité $\approx$ calculer M^m

Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses et produit

- Sensibilité $\approx$ calculer M^m
- Sensibilité $\approx$ calculer $M_1 \times M_2 \times \cdots M_m$

Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses et produit

- Sensibilité $\approx$ calculer M^m
- Sensibilité $\approx$ calculer $M_1 \times M_2 \times \cdots M_m$ et quelquefois ...

Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses et produit

- Sensibilité $\approx$ calculer M^m
- Sensibilité $\approx$ calculer $M_1 \times M_2 \times \cdots M_m$ et quelquefois . . .
- Sensibilité $\approx$ calculer **tous les** $M_i \times M_{i+1} \times \cdots M_{i+l}$ $(\forall i \in [1..m-l])$

Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses et produit

- Sensibilité $\approx$ calculer M^m
- Sensibilité $\approx$ calculer $M_1 \times M_2 \times \cdots M_m$ et quelquefois ...
- Sensibilité $\approx$ calculer **tous les** $M_i \times M_{i+1} \times \cdots M_{i+l}$ $(\forall i \in [1..m-l])$

et cela fait beaucoup de $\times$

OPTIMAL PREPROCESSING FOR ANSWERING ON-LINE PRODUCT QUERIES

Noga Alon †

Department of Mathematics
School of Mathematical Sciences
Tel Aviv University
Tel Aviv, Israel 69978

Baruch Schieber ‡

Department of Computer Science
School of Mathematical Sciences
Tel Aviv University
Tel Aviv, Israel 69978

ABSTRACT

We examine the amount of preprocessing needed for answering certain on-line queries as fast as possible. We start with the following basic problem. Suppose we are given a semigroup $(S,\circ)$. Let $s_1, \ldots, s_n$ be elements of S. We want to answer on-line queries of the form, "What is the product $s_i \circ s_{i+1} \circ \cdots \circ s_{j-1} \circ s_j$?" for any given $1 \le i \le j \le n$. We show that a preprocessing of $\Theta(n\lambda(k,n))$ time and space is both necessary and sufficient to answer each such query in at most k steps, for any fixed k. The function $\lambda(k,.)$ is the inverse of a certain function at the $\lfloor k/2 \rfloor$–th level of the primitive recursive hierarchy. In case linear preprocessing is desired, we show that one can answer each such query in at most $O(\alpha(n))$ steps and that this is best possible. The function $\alpha(n)$ is the inverse Ackermann function.

We also consider the following extended problem. Let T be a tree with an element of S associated with each of its vertices. We want to answer on-line queries of the form, "What is the product of the elements associated with the vertices along the path from u to v?" for any pair of vertices u and v in T. We derive results, which are similar to the above, for the preprocessing needed for answering such queries.

All our sequential preprocessing algorithms can be parallelized efficiently to give optimal parallel algorithms which run in $O(\log n)$ time on a CREW PRAM. These parallel algorithms are optimal in both running time and total number of operations.

† The research of this author was supported in part by Allon Fellowship, Bat Sheva de Rothschild Foundation and the Fund for Basic Research administered by the Israel Academy of Sciences.

‡ The research of this author was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under contract number DE-AC02-76ER03077.

We examine the amount of preprocessing needed for answering certain on-line queries as fast as possible. We start with the following basic problem. Suppose we are given a semigroup $(S, \circ)$. Let $s_1, \ldots, s_n$ be elements of S. We want to answer on-line queries of the form, "What is the product $s_i \circ s_{i+1} \circ \cdots \circ s_{j-1} \circ s_j$?" for any given $1 \leq i \leq j \leq n$. We show that a preprocessing of $\Theta(n\lambda(k, n))$ time and space is both necessary and sufficient to answer each such query in at most k steps, for any fixed k. The function $\lambda(k, .)$ is the inverse of a certain function at the $\lfloor k/2 \rfloor$–th level of the primitive recursive hierarchy. In case linear preprocessing is desired, we show that one can answer each such query in at most $O(\alpha(n))$ steps and that this is best possible. The function $\alpha(n)$ is the inverse Ackermann function.

We examine the amount of preprocessing needed for answering certain on-line queries as fast as possible. We start with the following basic problem. Suppose we are given a semigroup $(S,\circ)$. Let $s_1, \ldots, s_n$ be elements of S. We want to answer on-line queries of the form, "What is the product $s_i \circ s_{i+1} \circ \cdots \circ s_{j-1} \circ s_j$?" for any given $1 \le i \le j \le n$. We show that a preprocessing of $\Theta(n\,\lambda(k,n))$ time and space is both necessary and sufficient to answer each such query in at most k steps, for any fixed k. The function $\lambda(k,.)$ is the inverse of a certain function at the $\lfloor k/2 \rfloor$–th level of the primitive recursive hierarchy. In case linear preprocessing is desired, we show that one can answer each such query in at most $O(\alpha(n))$ steps and that this is best possible. The function $\alpha(n)$ is the inverse Ackermann function.

Si $\leq k$ multiplications pour calculer *Query(i,j)* ...

We examine the amount of preprocessing needed for answering certain on-line queries as fast as possible. We start with the following basic problem. Suppose we are given a semigroup $(S,\circ)$. Let $s_1, \ldots, s_n$ be elements of S. We want to answer on-line queries of the form, "What is the product $s_i \circ s_{i+1} \circ \cdots \circ s_{j-1} \circ s_j$?" for any given $1 \le i \le j \le n$. We show that a preprocessing of $\Theta(n\lambda(k,n))$ time and space is both necessary and sufficient to answer each such query in at most k steps, for any fixed k. The function $\lambda(k,.)$ is the inverse of a certain function at the $\lfloor k/2 \rfloor$–th level of the primitive recursive hierarchy. In case linear preprocessing is desired, we show that one can answer each such query in at most $O(\alpha(n))$ steps and that this is best possible. The function $\alpha(n)$ is the inverse Ackermann function.

Si $\leq k$ multiplications pour calculer *Query(i,j)* ...

alors *Preprocessing* en $\Theta\big(n\lambda(k,n)\big)$

We examine the amount of preprocessing needed for answering certain on-line queries as fast as possible. We start with the following basic problem. Suppose we are given a semigroup $(S,\circ)$. Let $s_1, \ldots, s_n$ be elements of S. We want to answer on-line queries of the form, "What is the product $s_i \circ s_{i+1} \circ \cdots \circ s_{j-1} \circ s_j$?" for any given $1 \le i \le j \le n$. We show that a preprocessing of $\Theta(n\,\lambda(k,n))$ time and space is both necessary and sufficient to answer each such query in at most k steps, for any fixed k. The function $\lambda(k,.)$ is the inverse of a certain function at the $\lfloor k/2 \rfloor$–th level of the primitive recursive hierarchy. In case linear preprocessing is desired, we show that one can answer each such query in at most $O(\alpha(n))$ steps and that this is best possible. The function $\alpha(n)$ is the inverse Ackermann function.

Si $\le k$ multiplications pour calculer *Query(i,j)* ...

alors *Preprocessing* en $\Theta\big(n\lambda(k,n)\big)$

$\lambda(0,n) = \lceil \frac{n}{2} \rceil \quad \lambda(1,n) = \lceil \sqrt{n} \rceil \quad \lambda(2,n) = \log(n) \quad \lambda(3,n) = \log\log(n) \quad \lambda(4,n) = \log^*(n)$

We examine the amount of preprocessing needed for answering certain on-line queries as fast as possible. We start with the following basic problem. Suppose we are given a semigroup $(S,\circ)$. Let $s_1, \ldots, s_n$ be elements of S. We want to answer on-line queries of the form, "What is the product $s_i \circ s_{i+1} \circ \cdots \circ s_{j-1} \circ s_j$?" for any given $1 \le i \le j \le n$. We show that a preprocessing of $\Theta(n\lambda(k,n))$ time and space is both necessary and sufficient to answer each such query in at most k steps, for any fixed k. The function $\lambda(k,.)$ is the inverse of a certain function at the $\lfloor k/2 \rfloor$–th level of the primitive recursive hierarchy. In case linear preprocessing is desired, we show that one can answer each such query in at most $O(\alpha(n))$ steps and that this is best possible. The function $\alpha(n)$ is the inverse Ackermann function.

Si $\le k$ multiplications pour calculer *Query(i,j)* ...

alors *Preprocessing* en $\Theta\big(n\lambda(k,n)\big)$

$\lambda(0,n) = \lceil \frac{n}{2} \rceil \quad \lambda(1,n) = \lceil \sqrt{n} \rceil \quad \lambda(2,n) = \log(n) \quad \lambda(3,n) = \log\log(n) \quad \lambda(4,n) = \log^*(n)$

$u_i = u_{i-1} + 2 \qquad u_i = u_{i-1} + 1 + 2\sqrt{u_{i-1}} \qquad u_i = u_{i-1} \times 2 \qquad u_i = u_{i-1}^2 \qquad u_i = 2^{u_i - 1}$

Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses et produit

- déduit de [Alon Schieber 87] : si $|i-j| \geq \sqrt{m}$, $Query(i,j)$ calculable en au plus 2 multiplications après un preprocessing linéaire

Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses et produit

- déduit de [Alon Schieber 87] : si $|i-j| \geq \sqrt{m}$, $Query(i,j)$ calculable en au plus 2 multiplications après un preprocessing linéaire
- mais notre problème est "plus facile" :
  Sensibilité $\approx$ calculer **les** $M_i \times M_{i+1} \times \cdots \times M_{i+l}$ $(\forall i \in [1..m-l])$

Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses et produit

- déduit de [Alon Schieber 87] : si $|i-j| \geq \sqrt{m}$, $Query(i,j)$ calculable en au plus 2 multiplications après un preprocessing linéaire
- mais notre problème est "plus facile" :
  Sensibilité $\approx$ calculer **les** $M_i \times M_{i+1} \times \cdots \times M_{i+l}$ $(\forall i \in [1..m-l])$
- nombre de multiplications amorti constant pour chaque i successif.

Calculer la sensibilité d'une graine

implémentation pratique : matrices creuses et produit

- déduit de [Alon Schieber 87] : si $|i-j| \geq \sqrt{m}$, $Query(i,j)$ calculable en au plus 2 multiplications après un preprocessing linéaire
- mais notre problème est "plus facile" :
  Sensibilité $\approx$ calculer **les** $M_i \times M_{i+1} \times \cdots \times M_{i+l}$ $(\forall i \in [1..m-l])$
- nombre de multiplications amorti constant pour chaque i successif.
- mais matrices "pleines" assez rapidement (graines normales), à priori moins gênant pour graines positionnées ...

Merci pour les nombreux PJIs encadrées cette année !!
n'hesitez pas à en proposer encore plus l'année prochaine :-)