UE d'informatique :
Automates et langages formels en informatique musicale

Marc Chemillier

Master M2 Atiam (Ircam), 2014-2015

Algorithme de Morris & Pratt (logiciel d'improvisation ImproteK)

Oracle des facteurs (logiciels d'improvisation OMax, ImproteK)

- Construction de l'oracle
    - Liens suffixiels
    - Utilisation musicale de l'oracle

1. Algorithme de Morris & Pratt

1.1 Problème du pattern matching

Exemple : Rechercher le motif *x* = *aaaab* dans le texte *t* = *aaaacaaaab*

a a a a ***c*** *a a a a b*
a a a a ***b*** **échec !!!**
a a a ***a*** **échec !!!**
a a ***a*** **échec !!!**

-> on se décale à chaque fois ***d'une lettre dans t***, car chaque lettre est potentiellement le début du motif *x*

problème : on n'utilise pas toute l'information obtenue par la lecture de *t* conduisant à l'échec

a a a a ***c*** *a a a a b*
a a a a ***b*** **échec !!! information : à cause de la lettre *c* dans le texte *t***
-> or le motif *x* ne contient pas la lettre *c*
=> on peut se décaler ***de toutes les lettres jusqu'à c incluse***

a a a a ***b*** motif trouvé !!!

algorithme de Morris & Pratt : utiliser toute l'information contenue dans le motif x pour optimiser les décalages

-> c'est l'automate de $\Sigma^* x$ qui fait ça !...

utilisation musicale :

- pattern matching sur des grilles de jazz (ImproteK) : on adapte des fragments de solos joués sur une grille à la grille d'un autre morceau **variante de Morris & Pratt** (on ne recherche pas un motifs fixe, mais des facteurs communs)
- recombinaison de solos en mode free (OMax) : basé sur une variante de l'algorithme de Morris & Pratt appelée **oracle des facteurs**

1.2 Utilisation d'une fonction d'échec

Algorithme de Morris & Pratt : calcul direct de l'AFD pour $\Sigma^* x$, **sans passer par la déterminisation de l'AFN**. On utilise une fonction d'échec définie sur les états p de l'écorché de x. Soit w le mot lu de l'état 0 à l'état p :

$f(p)$ = état d'arrivée du plus long suffixe propre de w qui est aussi préfixe de w (donc de x)

Exemple : Fonction d'échec pour l'écorché de $x = bacbb$:

État p	0	1	2	3	4	5
$f(p)$		0	0	0	1	1

• pour $p = 2$, le mot lu est ba, il n'a pas de suffixe propre qui soit préfixe, donc $f(2) = 0$

• pour $p = 4$, le mot lu est $bacb$, le plus long suffixe propre qui soit préfixe est b, donc $f(4) = 1$

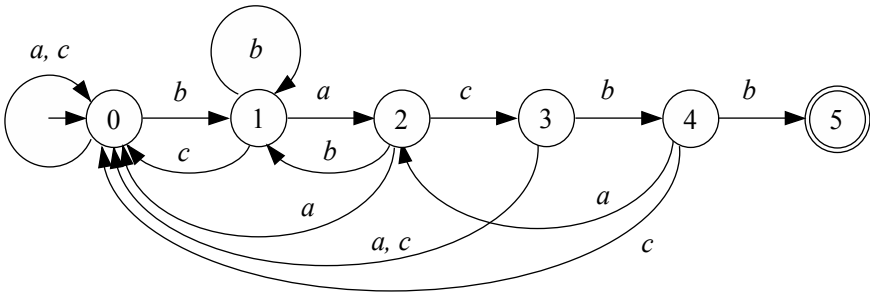
• pour $p = 5$, le mot lu est $bacbb$, le plus long suffixe propre qui soit préfixe est b, donc $f(5) = 1$

On ajoute de nouvelles transitions dans l'écorché de x de la manière suivante :

pour toute lettre a telle que $\delta(p, a)$ non défini, on pose

• $\delta(p, a) = \delta(f(p), a)$ si $p \neq 0$,

• $\delta(0, a) = 0$.



Exemple : Lecture de l'AFD pour chercher le motif x = *bacbb* dans le texte t = *abacbacbba*

a b a c ***b a c b b*** *a*
0 1 2 3 4 2 3 4 **5** motif trouvé !!!

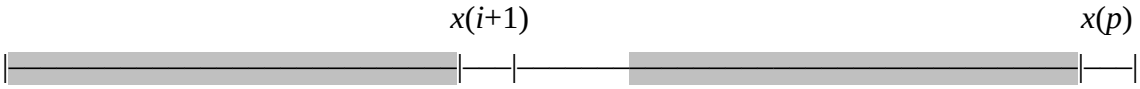
L'AFD du digicode ci-dessus correspond aux mots se terminant par x (langage $\Sigma^* x$) :
il détecte la **première occurrence** de x dans t
Dans la recherche de motif, on s'intéresse en général à toutes les occurrences de x dans t
-> il faut compléter l'AFD ci-dessus avec des transitions partant de 5

1.3 Algorithme de calcul de la fonction d'échec

Pour calculer facilement $f(p)$, il faut voir si on peut le déduire de $f(p-1)$.
-> Il se trouve que ça marche.

On note le motif $x = x(1)\ldots x(n)$
On suppose construits $f(1)$, $f(2)\ldots$ $f(p-1)$ et on cherche $f(p)$.

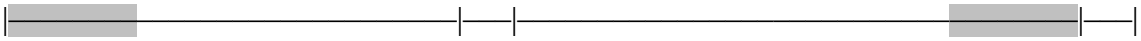
$i = f(p-1)$



• si la lettre $x(i+1) = x(p)$
alors $x(1)\ldots x(i+1)$ est à la fois préfixe et suffixe,
donc on peut prolonger le suffixe précédent : $f(p) = i + 1 = f(p-1) + 1$

• si la lettre est différente, il faut essayer un suffixe plus court, lequel ?
i = ?

$x(p)$



=> l'astuce consiste à remarquer qu'il doit être <u>suffixe du suffixe précédent :</u>

donc <u>on réapplique *f* :</u>

$i = f(f(p-1))$

et on recommence la comparaison entre $x(i+1)$ et $x(p)$

<u>calcul de la fonction d'échec *f*</u>

```
f(0) ← –1
i ← –1
pour p = 1 à n
        tant que i≥0 et x(i+1) ≠ x(p)
                i ← f(i)
        i ← i+1
        f(p) ← i
```

En fait, l'algorithme de Morris & Pratt ne construit pas explicitement l'AFD de $\Sigma^* x$. Il se contente de lire le texte *t* en avançant dans l'écorché de *x* (AFD), et en effectuant les retours arrière définis par la fonction d'échec.

```
p ← 0 (état initial)
tant que t non vide
avancer dans t, a ← 1ère lettre de t
        tant que p  0 et δ(p, a) non défini
                p ← f(p)
                    si δ(p, a) défini alors avancer lecture p ← δ(p, a)
        si p ∈ F alors signaler occurrence de x dans t
```

L'algorithme naïf de détection de motif consiste à essayer de lire *x* dans le texte à partir de la 1ère lettre, puis de la 2ème, puis de la 3ème, etc.

- en se décalant à chaque fois **<u>d'une seule lettre</u>**,
- **<u>en relisant *ce qu'on a déjà lu*</u>** si un préfixe de *x* commençait à la lettre précédente.

Dans l'algorithme de Morris & Pratt, **<u>on ne relit jamais *ce qu'on a déjà lu*</u>**

-> c'est cela qui accélère considérablement le calcul.

Le cas le plus défavorable pour l'algorithme naïf est celui où

- on lit la quasi-intégralité du motif jusqu'à l'avant-dernière lettre
- échec à la dernière lettre, on se décale d'une lettre *en relisant ce qui est déjà lu*

-> avec Morris & Pratt, on avance toujours ***de façon linéaire dans l'AFD*** sans revenir en arrière.

Exemple 1 : motif $x = a^{n+1}$ dans le texte $t = (a^n b)^m$

• échec du b après avoir lu a^n, la lettre b n'est pas dans le motif, donc on a $\delta(n, b) = 0$

-> c'est le cas de l'exemple initial : **on se décale de la longueur du motif** (jusqu'à b inclus)

Exemple 2 : motif $x = a^n b$ dans le texte $t = a^{(n+1)m}$

• échec du a après avoir lu a^n, on a une boucle $\delta(n, a) = n$, donc on reste dans le même état, ce qui revient à se décaler d'une lettre, **mais sans relire le préfixe a^n**

Morris & Pratt est beaucoup plus efficace que l'algorithme naïf dans les cas où il y a de nombreuses occurrences du motif (ou de ses préfixes) dans le texte.

S'il y a très peu d'occurrences du motif et de ses préfixes (par exemple si la première lettre du motif n'apparaît jamais dans le texte), les deux algorithmes sont équivalents.

2. Construction de l'oracle

2.1 Liens suffixiels

L'ensemble des facteurs d'un mot x est fini, donc reconnaissable par automate. On obtient facilement un AFN reconnaissant tous les facteurs en prenant l'écorché de x et en rendant tous les états initiaux et terminaux.

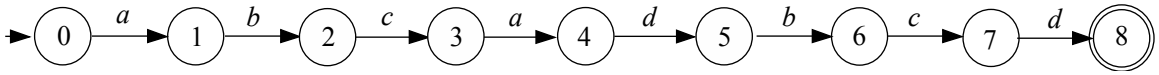
Problème : **il n'existe pas de construction simple pour l'AFD des facteurs de x** (contrairement à l'automate des mots se terminant par x).

L'« oracle des facteurs » a été introduit pour pallier cette difficulté.

L'*oracle des facteurs* est un automate qui reconnaît tous les facteurs d'un mot x, mais avec quelques mots en plus. Il est utilisé dans les applications de bio-informatique de façon **négative** : si un motif n'est pas reconnu par l'oracle d'une séquence ADN, on sait qu'il n'apparaît pas dans cette séquence.

On peut donner directement l'AFD correspondant (sans déterminisation) par une technique analogue à l'algorithme de Morris & Pratt.

Exemple : écorché de la séquence $x = abcadbcd$



Construction de l'oracle : on part de l'écorché de x, et on ajoute des transitions à l'aide d'une fonction f dite de « lien suffixiel » entre états, en construisant simultanément f et les transitions.

Au départ, on place un lien suffixiel de 1 vers 0. Puis on suppose l'oracle construit avec ses liens suffixiels jusqu'à l'état p compris. La lettre suivante a donne un nouvel état $\delta(p, a) = p+1$.

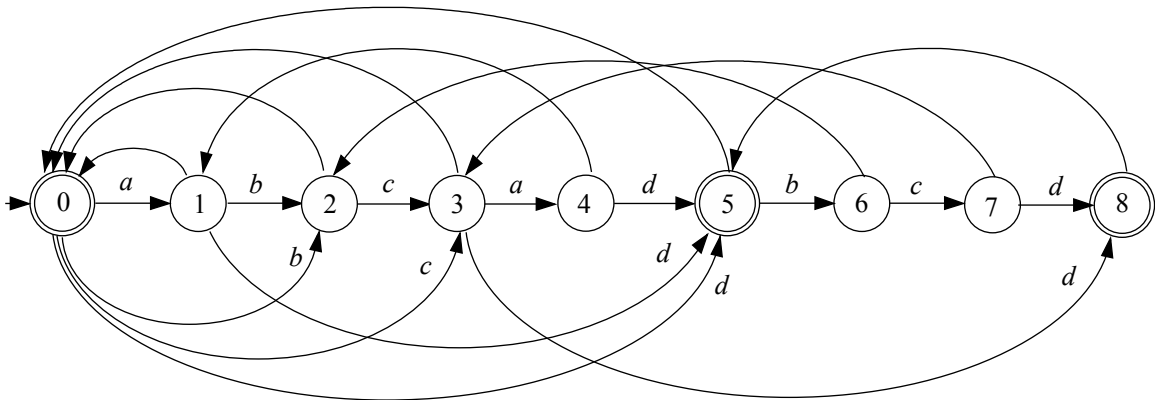
Pour ajouter les transitions, on suit les liens suffixiels déjà existants $p' = f(p)$, puis $p' = f(f(p))$, etc.

• si $\delta(p', a)$ est non défini, on ajoute une transition $\delta(p', a) = p+1$

- si $p' \neq 0$, on continue à suivre les liens,
- si $p' = 0$, on stoppe et on crée un lien suffixiel en posant $f(p+1) = 0$

• si $\delta(p', a)$ est défini, on stoppe (pas de nouvelle transition), et on crée un lien suffixiel en posant $f(p+1) = \delta(p', a)$ (= état d'arrivée de la transition existante)

Remarque : toutes les transitions arrivant dans un même état ont la même lettre.



• Oracle des facteurs : tous les états sont terminaux

• Oracle des suffixes : les états terminaux sont ceux du chemin suffixiel partant du dernier état, soit 8 (suffixe *cd*), 5 (suffixe *d*), 0 (suffixe vide)

Les flèches du dessus ***sans étiquette*** sont les liens suffixiels.

Les flèches du dessous avec étiquette sont les transitions additionnelles (ajoutées à l'écorché).

Tant qu'on lit des lettres nouvelles (*a*, *b*, *c*, ...), les liens suffixiels vont vers 0. Quand il y a une répétition (par ex. *a*), les liens suffixiels vont vers des états plus grands.

2.2 Utilisation musicale de l'oracle

Dans le logiciel d'improvisation OMax, on parcourt l'oracle en suivant les transitions, mais aussi en empruntant éventuellement les liens suffixiels. Cela revient à étendre l'oracle en lui ajoutant des transitions vides :

d b c — 1er motif prélevé = *a b c a* ***d b c*** *d*

5 6 7

On emprunte le lien suffixiel de 7 à 3 pour reprendre la lecture vers 4 :

d b c a d b — 2e motif prélevé = *a b c* ***a d b*** *c d*

(3) 4 5 6

Le 1er motif *dbc* se terminait par *bc*. Or le 2ème motif *adb* est **précédé** par *bc* dans la séquence.

1er motif prélevé = *a b c a* ***d b c*** *d*

2ème motif prélevé = *a* (*b c*) ***a d b*** *c d*

Dans la succession des 2 motifs *dbc* + *adb*, le 2ème prélèvement devient (*bc*)*adb* :

d b c a d b — 2e motif **réellement** prélevé = *a*(***b c***)***a d b*** *c d*

(2 3) 4 5 6

Ainsi les deux motifs réellement prélevés se **chevauchent** : *d*(*bc*) et (*bc*)*adb.*

C'est une propriété fondamentale de l'oracle :

Les dernières lettres lues avant de quitter un état par un lien suffixiel sont identiques à celles qui précèdent l'état d'arrivée du lien (ex : *bc* dans le parcours ci-dessus).

Ces lettres constituent une partie commune entre les motifs lus avant et après le lien suffixiel. Ainsi les liens suffixiels permettent d'enchaîner des motifs qui se chevauchent avec une partie commune (ex : *d*(*bc*) et (*bc*)*adb* dans le parcours ci-dessus).

Propriété fondamentale. *Si p est l'état d'arrivée d'un préfixe w de x, le lien suffixiel f(p) correspond à l'état d'arrivée du plus long suffixe de w qui est répété à gauche, c'est-à-dire qui est facteur non suffixe de w.*

Attention : Ça ressemble à la fonction de défaut de Morris & Pratt, mais petite différence...

• fonction de lien suffixiel :

f(*p*) = état d'arrivée du plus long suffixe propre de *w* qui est aussi **facteur** de *w* (donc de *x*)

• fonction de saut de Morris & Pratt :

f(*p*) = état d'arrivée du plus long suffixe propre de *w* qui est aussi **préfixe** de *w* (donc de *x*)

On peut améliorer le mode de navigation dans l'oracle (Assayag & Bloch 2007) :

avant de suivre un lien suffixiel, on regarde quelques états plus loin s'il n'y a pas un autre lien

suffixiel **avec un suffixe propre répété plus long**

-> cela permet d'enchaîner des motifs avec des chevauchements plus grands

Références

• oracle des suffixes (simulation stylistique)

Allauzen, Cyril & Maxime Crochemore, Mathieu Raffinot, Factor oracle : A new structure for pattern matching, *SOFSEM '99*: *Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science, Springer-Verlag, 1999, p. 291-306 (biblio).

Assayag, Gérard, Shlomo Dubnov, Olivier Delerue, Guessing the Composer's Mind: Applying Universal Prediction to Musical Style, *Proceedings of the ICMC (Int. Computer Music Conf.)*, 1999, p. 496-499 (biblio).

Assayag, Gérard, Shlomo Dubnov, Using factor oracles for machine improvisation, *Soft Computing*, special issue on Formal Systems and Music, G. Assayag, V. Cafagna, M. Chemillier (eds.), 8 (9) (2004) 604–610 (biblio).

Assayag, Gérard, Georges Bloch, Navigating the oracle, *Proceedings of the ICMC (Int. Computer Music Conf.)*, 2007 (biblio).