DES BORNES EXACTES DE LA COMPLEXITE DES ALGORITHMES SEQUENTIELS DE RECHERCHE D'UN MOTIF

Christophe Hancart

Résumé

Dans cet article, nous étudions la complexité exacte du problème du calcul de toutes les occurrences d'un motif dans un texte par des algorithmes qui, comme celui de Knuth, Morris et Pratt, n'ont accès au texte qu'au travers une fenêtre glissante de longueur 1, en supposant que l'alphabet est inconnu des algorithmes et contient au moins deux lettres.

Nous donnons une famille d'algorithmes linéaires en temps et en espace qui réalisent le calcul en effectuant au plus $\lfloor (2 - \frac{1}{m})\, n \rfloor$ comparaisons entre caractères du motif et caractères du texte, et en comparant chaque caractère du texte au plus $\min\{1 + \lfloor \log_2 m \rfloor, \mathrm{card}(A)\}$ fois, où m est la longueur du motif, n celle du texte et A l'alphabet. Nous montrons que ces bornes sont les bornes exactes du problème. Les algorithmes optimaux résultent d'une implémentation d'automates finis.

Abstract

String matching is the problem of finding all the occurrences of a string, called the pattern, within another, called the text. We give sharp bounds when only one head moving from the left to the right of the text is used, for the general case where the alphabet is unknown to the algorithms : $\lfloor (2 - \frac{1}{m})\, n \rfloor$ comparisons between pattern characters and text characters, and $\min\{1 + \lfloor \log_2 m \rfloor, \mathrm{card}(A)\}$ comparisons on each text character, where m is pattern length, n is text length, and A is the alphabet. These results improve the previous results given by Knuth, Morris and Pratt, and, more recently, by Simon.

1 Introduction

Le problème de la *recherche d'un motif* consiste à localiser toutes les occurrences d'un mot x, appelé le *motif*, de longueur non nulle m dans un mot y, appelé le *texte*, de longueur n, où les deux mots sont considérés sur un alphabet donné A.

Received by the editors April 1993, revised February 1994.
Communicated by M. Boffa.
AMS Mathematics Subject Classification : 68Q20, 68Q25, 68R15, 68U15.
Keywords : string matching, string searching, text editing, computational complexity, worst case behavior, finite automata.

Nous disons d'un algorithme de recherche d'un motif qu'il est *séquentiel* s'il n'a accès au texte qu'au travers une fenêtre glissante de longueur 1 et s'il ne prend connaissance de la longueur du texte qu'une fois tout le texte lu.

Dans cet article, nous étudions la complexité exacte des algorithmes séquentiels de recherche d'un motif dans le cas où l'alphabet A est quelconque, *inconnu* des algorithmes, et qu'il contient au moins deux lettres. Pour de tels algorithmes, le seul moyen d'obtenir des informations sur le caractère courant du texte est de comparer ce caractère à des lettres du motif. La complexité s'exprime ainsi naturellement en nombre de comparaisons (on sous-entend : entre caractères du motif et caractères du texte). Typiquement, on donne le *nombre total de comparaisons* effectuées dans le pire des cas et le *délai*, c'est à dire le nombre de comparaisons faites sur un même caractère de texte dans le pire des cas.

Remarquons toutefois que lorsque l'alphabet est connu à l'avance de l'algorithme et lorsque sa taille n'est pas trop importante, l'automate minimal déterministe $\mathcal{A}(x)$ qui reconnaît l'ensemble des mots qui se terminent par x peut être implémenté (voir [1, 11, 5]). L'avantage de cette solution est de permettre une recherche en temps réel (l'automate traite exactement un caractère de texte par unité de temps), et son inconvénient sa dépendance vis-à-vis de l'alphabet.

Le plus connu des algorithmes séquentiels de recherche d'un motif de complexité indépendante de A est l'algorithme KMP de Knuth, Morris et Pratt [10]. Il nécessite un espace supplémentaire $O(m)$ et s'exécute en temps $O(n+m)$. Le nombre de total comparaisons est d'au plus $2n-1$ et le délai n'excède jamais $\lfloor \log_\Phi(m+1) \rfloor$, où $\Phi = \frac{1+\sqrt{5}}{2}$ est le nombre d'or (voir [10, 5]).

I. Simon (voir [12, 2, 5, 9, 13][1]) a proposé récemment un algorithme, noté ici MPS, qui peut être vu comme un compromis entre l'implémentation par automate et l'algorithme KMP. Il considère l'automate $\mathcal{A}(x)$ et remarque que les flèches qui ne reviennent pas à l'état initial sont au nombre d'au plus $2m$. Si l'on ne retient que ces flèches *significatives*, on est ramené à un algorithme à espace supplémentaire $O(m)$. Une transition n'est maintenant plus calculée comme un simple branchement, mais au moyen de comparaisons du caractère du texte en cours d'analyse avec les étiquettes de différentes flèches significatives issues de l'état courant. Toujours est-il que ces flèches peuvent calculées en temps $O(m)$, et qu'elles peuvent examinées dans un *ordre* tel que la recherche soit au moins aussi efficace que celle de l'algorithme KMP. Le pire des cas est toujours en $2n-1$, mais le délai est maintenant majoré par $\min\{\lfloor \log_\Phi(m+1) \rfloor, \mathrm{card}(A)\}$.

Nous notons $\mathcal{SA}(x)$ l'ensemble des algorithmes qui simulent l'automate $\mathcal{A}(x)$ en comparant le caractère courant du texte aux étiquettes de flèches significatives issues de l'état courant sans effectuer deux fois la même comparaison. Les algorithmes de $\mathcal{SA}(x)$ sont des algorithmes séquentiels de recherche du motif x, et l'algorithme MPS est l'un d'entre eux.

Nous étendons le travail de Simon en étudiant la complexité des algorithmes de $\mathcal{SA}(x)$. Nous prouvons que :

(i) Le nombre total de comparaisons effectuées par les algorithmes de $\mathcal{SA}(x)$ est d'au plus $2n-1$. Il n'est donc pas nécessaire d'examiner les flèches significa-

[1] La remarque de Simon date de 1989. Le papier dans lequel il la relate ne date lui que de 1993.

tives dans l'ordre utilisé dans l'algorithme MPS pour obtenir un algorithme de complexité linéaire.

(ii) Le délai des algorithmes de $\mathcal{SA}(x)$ égale au plus $\min\{1 + \lfloor \log_2 m \rfloor, \mathrm{card}(A)\}$. Nous améliorons ainsi la borne supérieure du délai obtenue par Simon pour les algorithmes de notre problème, et établissons du même coup le délai exact de l'algorithme MPS.

Nous améliorons aussi la borne supérieure du nombre total de comparaisons effectuées en montrant que :

(iii) Le nombre total de comparaisons effectuées par les algorithmes d'une certaine classe de $\mathcal{SA}(x)$ que nous préciserons, est d'au plus $\lfloor (2 - \frac{1}{m})\, n \rfloor$.

Enfin, nous prouvons que les bornes supérieures annoncées en (ii) et (iii) sont les bornes exactes du problème :

(iv) N'importe quel algorithme séquentiel de recherche d'un motif effectue au total au moins $\lfloor (2 - \frac{1}{m})\, n \rfloor$ comparaisons dans le pire des cas, et a un délai au moins égal à $\min\{1 + \lfloor \log_2 m \rfloor, \mathrm{card}(A)\}$ dans le pire des cas.

Rappelons que pour le problème plus étudié des algorithmes à fenêtre glissante de taille égale à la longueur du motif, des bornes exactes ne sont (actuellement) connues que lorsque la longueur du motif est soit petite, soit infinie (voir [7, 8, 4]). Ajoutons encore que lorsque la longueur du motif est grande, la complexité de ce problème est voisine de n comparaisons, alors qu'elle est proche de $2n$ pour le nôtre.

Le reste de cet article est organisé comme suit. Dans la section 2, nous fixons les notations et nous exposons l'idée générale de notre analyse. Dans la section 3, nous proposons un comptage des flèches significatives qui précise celui de Simon. Ce qui nous permet, dans la section 4, d'établir les complexités annoncées en (i), (ii) et (iii). Dans la section 5, nous établissons les bornes inférieures annoncées en (iv). La dernière section contient nos conclusions.

2 Préliminaires

L'alphabet A étant donné, on désigne par A^* l'ensemble des mots sur A. La longueur d'un mot $u \in A^*$ est notée $|u|$. Le mot de longueur nulle est noté ε. L'ensemble des mots sur A de longueur non nulle est noté A^+. Pour tout mot $u \in A^*$, on note $u[i]$ le i-ème caractère de u et $u[i \ldots j]$ le mot $u[i]\, u[i+1] \cdots u[j]$. On dit d'un mot $v \in A^*$ qu'il est *un bord* du mot $u \in A^*$ si v est à la fois préfixe et suffixe de u ; si en plus v est différent de u, on dit que v est *un bord propre* de u ; et enfin si v est un bord propre de u de longueur maximale, on dit que v est *le bord* de u. On note Bord l'application qui à tout mot de longueur non nulle associe son bord. Pour tout prédicat p, $\mathbf{1}\{p\}$ vaut 1 si p est vrai, et 0 sinon.

Le problème de la recherche d'un motif $u \in A^*$ dans un texte $v \in A^*$ consiste à calculer toutes les positions j, $1 \leq j \leq |v| - |u| + 1$, telles que $v[j \ldots j + |u| - 1] = u$. De manière équivalente, celà revient à calculer l'ensemble des préfixe s de v qui

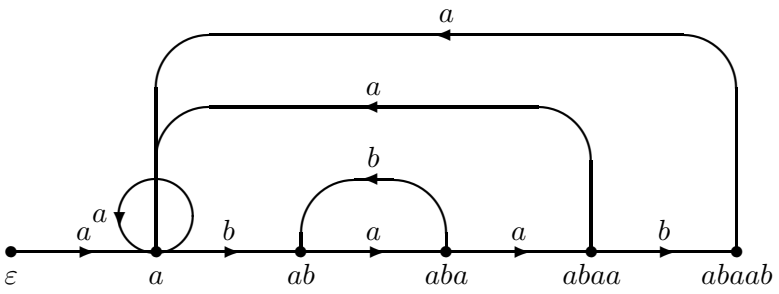

Figure 1: L'automate $\mathcal{A}(abaab)$, avec $a, b \in A$ et $a \neq b$, sans ses flèches nulles.

sont aussi des éléments de A^*u, l'ensemble des mots de A^* qui se terminent par u. L'ensemble A^*u étant un langage rationnel, il est reconnu par un automate.

L'automate minimal déterministe qui reconnaît A^*u est noté $\mathcal{A}(u)$. Ses états sont exactement les $|u| + 1$ préfixes de u, l'état initial étant ε et l'état terminal u. Les flèches de l'automate sont les triplets (p, a, q) de $A^* \times A \times A^*$ tels que q soit le plus long préfixe de u qui soit aussi suffixe de pa. L'ensemble des flèches de $\mathcal{A}(u)$ se calcule par induction sur la longueur des préfixes de u selon la méthode rappelée dans la proposition suivante :

Proposition 2.1 *(Folklore ; voir [1, 11, 5].) L'ensemble des flèches de $\mathcal{A}(\varepsilon)$ est vide. Ensuite, soient un mot $v \in A^*$ et une lettre $b \in A$. Soit w la cible de la flèche de $\mathcal{A}(v)$ issue de l'état v et étiquetée par la lettre b. Alors $w = \mathrm{Bord}(vb)$. Par conséquent, l'ensemble F_{vb} des flèches de $\mathcal{A}(vb)$ s'obtient de l'ensemble F_v des flèches de $\mathcal{A}(v)$ selon la formule :*

$$F_{vb} = F' \cup \{\, (vb, a, q) \mid (w, a, q) \in F' \,\},$$

où

$$F' = (F_v \setminus \{(v, b, w)\}) \cup \{(v, b, vb)\}.$$

Dans la terminologie employée par Simon, une flèche (p, a, q) de $\mathcal{A}(u)$ est dite *significative* si $q \neq \varepsilon$, et *nulle* sinon ; une flèche significative (p, a, q) est dite *avant* si $q = pa$, et *arrière* sinon. La figure 1 montre une représentation de l'automate $\mathcal{A}(u)$ pour le mot $u = abaab$, avec $a, b \in A$ et $a \neq b$, sans ses flèches nulles ; on compte 5 ($= |u|$) flèches avant, 4 flèches arrière, et donc $(5 + 1) \times \mathrm{card}(A) - (5 + 4)$ flèches nulles.

Pour toute la suite, m est une longueur non nulle de motif, $x \in A^+$ un motif de longueur m, n une longueur de texte et $y \in A^*$ un texte de longueur n.

Rappelons que nous notons par $\mathcal{SA}(x)$ l'ensemble des algorithmes qui simulent $\mathcal{A}(x)$ à partir de ses seules flèches significatives. Si p est l'état courant et a le caractère courant du texte, de tels algorithmes cherchent (chacun selon sa stratégie) dans l'ensemble des flèches significatives issues de p, une flèche d'étiquette a, en procédant par comparaisons successives des étiquettes (qui sont des lettres du motif x) avec le caractère a, et en ne considérant jamais deux fois la même flèche ; si

– Phase d'initialisation.
$csig[0] \leftarrow \{\}$
pour i de 1 à m **faire**
    $j \leftarrow \max(\{0\} \cup \{\, k \mid k \in csig[i-1] \text{ et } x[k] = x[i] \,\})$
    – Ici, $x[1 \ldots j]$ est le bord de $x[1 \ldots i]$.
    si $j \neq 0$ **alors**
        $csig[i-1] \leftarrow (csig[i-1] \setminus \{j\}) \cup \{i\}$
    sinon
        $csig[i-1] \leftarrow csig[i-1] \cup \{i\}$
    $csig[i] \leftarrow csig[j]$
fin pour
– Phase de recherche.
$i \leftarrow 0$
pour j de 1 à n **faire**
    $i \leftarrow \max(\{0\} \cup \{\, k \mid k \in csig[i] \text{ et } x[k] = y[j] \,\})$
    – Ici, $x[1 \ldots i]$ est le plus long préfixe de x qui soit aussi suffixe de $y[1 \ldots j]$.
    si $i = m$ **alors**
        Signaler une occurrence du motif x à la position $j - m + 1$.
fin pour

Figure 2: Schéma général des algorithmes de $\mathcal{SA}(x)$.

une telle flèche existe, disons (p, a, q), alors q est le nouvel état courant ; sinon le nouvel état courant est l'état initial, ε.

Un schéma général des algorithmes de $\mathcal{SA}(x)$ est proposé figure 2. Le tableau *csig* (pour : cibles significatives) est un tableau indicé de 0 à m. Pour tout $i \in \{0, \ldots, m\}$, l'élément $csig[i]$ du tableau *csig* est l'ensemble des longueurs des cibles des flèches significatives issues de l'état de longueur i ; autrement dit, $j \in csig[i]$ si et seulement si $(x[1 \ldots i], x[j], x[1 \ldots j])$ est une flèche significative de l'automate $\mathcal{A}(x)$. La première phase correspond au calcul du tableau *csig*, et la deuxième phase à la recherche du motif à proprement parler. La complexité des algorithmes de $\mathcal{SA}(x)$ sera établie dans la section 4.

Nous allons voir dans la suite de cette section qu'un décompte précis du nombre de flèches significatives de l'automate $\mathcal{A}(x)$ nous permettra d'obtenir des majorations des nombres de comparaisons effectuées par les algorithmes de $\mathcal{SA}(x)$ sur un caractère donné du texte comme sur tout le texte.

Si p est un préfixe du mot $u \in A^*$ et si q est un préfixe de p, on note $nfs_u(q \ldots p)$ le nombre de flèches significatives issues des états r de l'automate $\mathcal{A}(u)$ tels que q est préfixe de r et r préfixe de p ; on note plus simplement $nfs_u(p)$ à la place de $nfs_u(p \ldots p)$. On dit d'une suite $q, \ldots, p$ de préfixes consécutifs d'un mot $u \in A^*$ qu'elle est une *boucle* (on sous-entend : de l'automate $\mathcal{A}(u)$) s'il existe une lettre $a \in A$ telle que (p, a, q) soit une flèche arrière ou nulle de $\mathcal{A}(u)$. En reprenant l'exemple du mot $u = abaab$ de la figure 1, on a

$$nfs_u(\varepsilon) = nfs_u(ab) = nfs_u(abaab) = 1 \qquad \text{et} \qquad nfs_u(a \ldots abaa) = 7 \ ;$$

la suite a, ab, aba, $abaa$, de préfixes consécutifs de u est une boucle ; et la suite des

préfixes consécutifs ε, ..., *abaa* est aussi une boucle dès lors que l'alphabet contient au moins trois lettres.

Dans le pire des cas, le nombre de comparaisons effectuées dans un état donné de l'automate $\mathcal{A}(x)$ est égal au nombre de flèches significatives issues de cet état. Il en résulte la proposition suivante :

Proposition 2.2 *(Simon.) Le délai des algorithmes de $\mathcal{SA}(x)$ égale le maximum des nombres $nfs_x(p)$, où p décrit l'ensemble des préfixes de x. Il ne dépend pas de l'algorithme considéré dans $\mathcal{SA}(x)$.*

Pour obtenir des majorations du nombre total de comparaisons effectuées par des algorithmes de $\mathcal{SA}(x)$, on décompose le processus de recherche en boucles :

Proposition 2.3 *Supposons donné un algorithme de $\mathcal{SA}(x)$. Pour toute flèche (p, a, q) de $\mathcal{A}(x)$, notons $c(p, q)$ le nombre maximal de comparaisons effectuées par l'algorithme lors du calcul d'une transition de p à q ; et si (p, a, q) est une flèche arrière ou nulle, notons $c(q \dots p)$ la somme des nombres maximaux de comparaisons effectuées lors du calcul des transitions de q à p, puis d'une transition de p à q, soit*

$$c(q \dots p) = (\textstyle\sum_{i=|q|+1}^{|p|} c(q\,p[|q|+1 \dots i-1], q\,p[|q|+1 \dots i])) + c(p, q).$$

Notons encore C le nombre maximal de comparaisons effectuées par l'algorithme lors de la recherche du motif x dans le texte y. Alors il existe une suite de boucles, disons $((q_k, \dots, p_k))_{k\in\{1,\dots,t\}}$, telle que

$$q_t = \varepsilon, \qquad \textstyle\sum_{k=1}^{t}(|p_k| - |q_k| + 1) = n, \qquad \text{et} \qquad \sum_{k=1}^{t} c(q_k \dots p_k) \geq C.$$

Preuve. Examinons le cas non trivial où le texte n'est pas de longueur nulle.

Comme l'on cherche une majoration du nombre de comparaisons effectuées, on peut considérer, quitte à augmenter l'alphabet d'une lettre, que le dernier caractère du texte n'est pas une lettre du motif. Il s'en suit que la dernière transition calculée correspond à une flèche nulle, puis qu'au moins une boucle est exécutée.

Soit $R_1 = (r_l)_{l\in\{0,\dots,n\}}$ la suite des valeurs successives des états courants. Aux deux états r_i et r_j où i et j sont tels que $0 \leq i \leq j \leq n-1$ et que j soit l'entier minimal vérifiant $r_i = r_{j+1}$, correspond exactement la première boucle exécutée, à savoir la boucle $r_i, \dots, r_j$. Le nombre de comparaisons effectuées lors cette boucle est majoré par $c(r_i \dots r_j)$, et le nombre de caractères de texte lus est égal à $|r_j| - |r_i| + 1$. On pose alors $p_1 = r_j$ et $q_1 = r_i$, puis on réitère la décomposition en considérant la suite R_2 obtenue comme la juxtaposition des deux suites $(r_l)_{l\in\{0,\dots,i\}}$ et $(r_l)_{l\in\{j+2,\dots,n\}}$. Et ainsi de suite, jusqu'à obtenir une suite R_{t+1} réduite à ε. Ce qui termine la preuve de la proposition.

■

Ainsi, il suffira de connaître des majorations des nombres de flèches significatives $nfs_x(p)$ pour tous les préfixes p de x et $nfs_x(q \dots p)$ pour toutes les boucles $q, \dots, p$, pour obtenir des bornes des nombre de comparaisons effectuées. Un calcul de telles majorations est proposé dans la section suivante.

3 Comptage des flèches significatives de $\mathcal{A}(x)$

Dans cette section, nous établissons des bornes supérieures du nombre de flèches significatives issues d'un état quelconque de $\mathcal{A}(u)$ et du nombre de flèches significatives issues de chacun des états d'une boucle quelconque de $\mathcal{A}(u)$, pour tout mot $u \in A^*$.

Il résulte directement de la proposition 2.1 :

Lemme 3.1 *$nfs_\varepsilon(\varepsilon) = 0$. Soient un mot $u \in A^*$ et une lettre $a \in A$. Si v est un préfixe de ua, alors*

$$nfs_{ua}(v) = \begin{cases} nfs_u(v) + \mathbf{1}\{\mathrm{Bord}(ua) = \varepsilon\} & \text{si } v = u \\ nfs_u(\mathrm{Bord}(ua)) & \text{si } v = ua \\ nfs_u(v) & \text{sinon.} \end{cases}$$

Nous en déduisons la proposition suivante :

Proposition 3.2 *Soit un mot $u \in A^*$. Si (p, a, q) est une flèche arrière ou nulle de $\mathcal{A}(u)$, alors*

$$nfs_u(q \dots p) \leq 2\,|p| - 2\,|q| + \mathbf{1}\{q \neq \varepsilon\} + \mathbf{1}\{p \neq u\}.$$

Preuve. La proposition est vérifiée pour le mot de longueur nulle. Supposons la vraie pour le mot $v \in A^*$ et montrons qu'elle est vraie pour le mot vb, où $b \in A$.

Soit donc (p, a, q) une flèche arrière ou nulle de $\mathcal{A}(vb)$.

Si $\mathrm{Bord}(vb) = v$, c'est à dire si $v = b^{|v|}$, les flèches significatives de $\mathcal{A}(vb)$ sont exactement les $|v| + 1$ flèches avant et la flèche arrière (vb, b, vb) ; la proposition est donc vérifiée dans ce cas particulier.

Sinon $\mathrm{Bord}(vb)$ est un préfixe propre v. Si p est préfixe de v, la flèche (p, a, q) est aussi une flèche arrière ou nulle de $\mathcal{A}(v)$, et l'on déduit du lemme 3.1 que

$$nfs_{vb}(q \dots p) \leq nfs_v(q \dots p) + \mathbf{1}\{p = v\}.$$

Sinon $p = vb$. Posons $w = \mathrm{Bord}(vb)$. La flèche (w, a, q) est aussi une flèche de $\mathcal{A}(v)$ (voir figure 3). S'il s'agit d'une flèche avant, c'est à dire si $q = wa$, on écrit :

$$nfs_{vb}(q \dots p) = nfs_{vb}(q \dots v) + nfs_{vb}(vb)\ ;$$

d'où l'on déduit, en utilisant le lemme 3.1, que

$$nfs_{vb}(q \dots p) = nfs_v(w \dots v) + \mathbf{1}\{w = \varepsilon\}.$$

Sinon, il s'agit d'une flèche arrière ou nulle, et il vient de la même façon :

$$nfs_{vb}(q \dots p) = nfs_{vb}(q \dots w) + nfs_{vb}(w\,u[|w| + 1] \dots v) + nfs_{vb}(vb),$$

puis :

$$nfs_{vb}(q \dots p) = nfs_v(q \dots w) + nfs_v(w \dots v) + \mathbf{1}\{w = \varepsilon\}.$$

L'application de la proposition au mot v nous permet de conclure que

$$nfs_{vb}(q \dots p) \leq 2\,|p| - 2\,|q| + \mathbf{1}\{q \neq \varepsilon\} + \mathbf{1}\{p \neq vb\}$$

dans tous les cas où $\mathrm{Bord}(vb)$ est un préfixe propre v.

La proposition est donc vraie pour le mot vb. Ce qui achève la preuve.

■

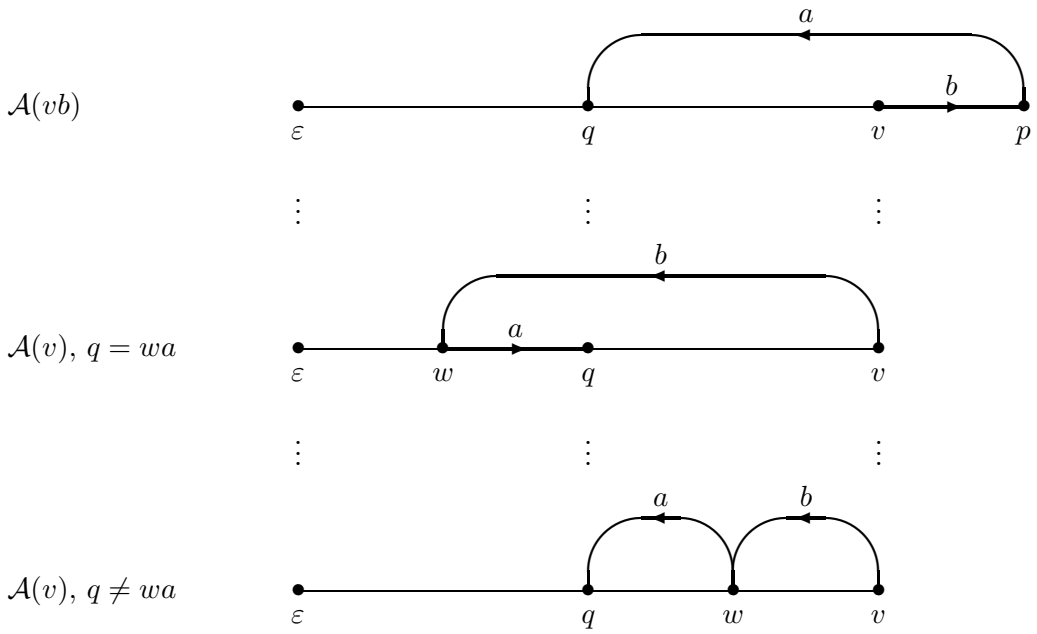

Figure 3: Preuve de la proposition 3.2.

Il résulte aussi directement de la proposition 2.1 le lemme suivant :

Lemme 3.3 *Soit un mot $u \in A^+$. Si v est un préfixe de u, alors*

$$nfs_u(v) = \begin{cases} 1 & \text{si } v = \varepsilon \\ nfs_u(\mathrm{Bord}(u)) & \text{si } v = u \\ nfs_u(\mathrm{Bord}(v)) + \mathbf{1}\{\mathrm{Bord}(v\,u[|v|+1]) = \varepsilon\} & \text{sinon.} \end{cases}$$

Il s'en déduit que :

Lemme 3.4 *Soit un mot $u \in A^+$. Si v est un préfixe de longueur non nulle de u tel que $2\,|\mathrm{Bord}(v)| \geq |v|$, alors $nfs_u(\mathrm{Bord}(v)) = nfs_u(\mathrm{Bord}^2(v))$.*

Preuve. Posons $w = v[1 \ldots 2\,|\mathrm{Bord}(v)| - |v|]$ et $a = v[|w|+1]$ (voir figure 4). Puisque wa est un bord propre de $\mathrm{Bord}(v)\,a$, le bord de $\mathrm{Bord}(v)\,a$ est de longueur non nulle. On applique alors le lemme 3.3 au préfixe propre $\mathrm{Bord}(v)$ de u.

■

Des deux lemmes précédents nous déduisons la proposition suivante :

Proposition 3.5 *Soit un mot $u \in A^+$. Si v est un préfixe u, alors*

$$nfs_u(v) \leq 1 + \lfloor \log_2 \min\{|v|+1,\ |u|\} \rfloor.$$

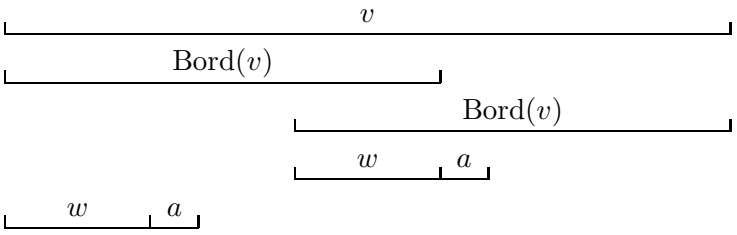

Figure 4: Preuve du lemme 3.4.

Preuve. D'après le lemme 3.3, il suffit de montrer que

$$nfs_u(v) \leq 1 + \lfloor \log_2(|v| + 1) \rfloor$$

pour chaque préfixe propre v de u.

Cette assertion est vérifiée pour le préfixe de longueur nulle. Supposons la vraie pour tout préfixe propre du préfixe propre v de u et montrons qu'elle est vraie pour v.

Soit i l'entier tel que

$$2^i \leq |v| + 1 < 2^{i+1},$$

et soit j l'entier tel que

$$|\mathrm{Bord}^{j+1}(v)| + 1 < 2^i \leq |\mathrm{Bord}^j(v)| + 1.$$

Pour tout $k \in \{0, \ldots, j-1\}$, on a :

$$2\,|\mathrm{Bord}^{k+1}(v)| \geq 2^{i+1} - 2 \geq |v| \geq |\mathrm{Bord}^k(v)|\ ;$$

d'où $nfs_u(\mathrm{Bord}^{k+1}(v)) = nfs_u(\mathrm{Bord}^{k+2}(v))$ d'après le lemme 3.4. On obtient ainsi que $nfs_u(\mathrm{Bord}(v)) = nfs_u(\mathrm{Bord}^{j+1}(v))$.

Or le lemme 3.3 entraîne $nfs_u(v) \leq nfs_u(\mathrm{Bord}(v)) + 1$; et l'assertion appliquée au mot $\mathrm{Bord}^{j+1}(v)$ entraîne $nfs_u(\mathrm{Bord}^{j+1}(v)) \leq i$.

Il en résulte que $nfs_u(v) \leq i + 1$. Ce qui valide l'assertion pour le mot v et termine la preuve.

■

4 Complexité des algorithmes de $\mathcal{SA}(x)$

Dans cette section, nous déduisons des propositions 2.2, 2.3, 3.2 et 3.5 démontrées dans les deux sections précédentes la complexité des algorithmes de $\mathcal{SA}(x)$.

Le résultat suivant a déjà été montré par Simon. Nous en donnons une autre preuve.

Proposition 4.1 *L'automate $\mathcal{A}(x)$ a au plus $2m$ flèche significatives. L'ensemble des flèches significatives de $\mathcal{A}(x)$ se calcule en temps $O(m)$.*

Preuve. Quitte à augmenter l'alphabet d'une lettre, on applique la proposition 3.2 à une flèche nulle issue de l'état terminal x ; d'où la borne supérieure de $2m$ pour le nombre total de flèches significatives de l'automate. Cette borne est atteinte dès que le mot x est de la forme ab^{m-1}, où a et b sont deux lettres distinctes de A.

D'après la proposition 2.1, le temps nécessaire au calcul des flèches significatives issues d'un état donné de l'automate est proportionnel à leur nombre. Le temps nécessaire au calcul de l'ensemble des flèches significatives est donc proportionnel à m.

■

Proposition 4.2 *Le nombre de comparaisons entre caractères du motif et caractères du texte Effectués par n'importe quel algorithme de $\mathcal{SA}(x)$ est inférieur à $2n-1$.*

Preuve. Soit (p, a, q) une flèche arrière ou nulle de $\mathcal{A}(x)$. Dans le pire des cas, le nombre de comparaisons exécutées lors de la boucle $q, \ldots, p$ égale $nfs_x(q \ldots p)$. D'après la proposition 3.2, ce nombre est majoré par $2\,(|p|-|q|+1)$ (respectivement par $2\,(|p|-|q|+1)-1$) si la flèche (p, a, q) est une flèche arrière (respectivement une flèche nulle).

Il résulte alors de la proposition 2.3 que le nombre de comparaisons effectuées durant la recherche est majoré par $2n-1$. Comme pour l'algorithme KMP, cette borne est atteinte lorsque $x = ab$ et $y = a^n$, avec $a, b \in A$ et $a \neq b$.

■

Il résulte des deux propositions précédentes la complexité des algorithmes de $\mathcal{SA}(x)$:

Théorème 4.3 *Les algorithmes de $\mathcal{SA}(x)$ (dont le schéma général est donné figure 2) peuvent être implantés en temps et en espace $O(m)$; leur phase de recherche s'exécute en temps $O(n)$.*

Pour la suite, on pose

$$\xi\colon A^* \to A^*$$

l'application définie par

$$\xi(\varepsilon) = \varepsilon, \qquad \xi(ua) = \xi(u)\,a\,\xi(u), \qquad \text{pour } u \in A^*,\ a \in A.$$

(Par exemple si a, b, c et d sont des lettres de A, $\xi(abcd) = abacabadabacaba$.)

Théorème 4.4 *Le délai de n'importe quel algorithme de $\mathcal{SA}(x)$ n'est jamais supérieur à $\min\{1 + \lfloor \log_2 m \rfloor, \operatorname{card}(A)\}$.*

Preuve. Le nombre de flèches significatives issues de chacun des états de l'automate $\mathcal{A}(x)$ est à la fois majoré par $1 + \lfloor \log_2 m \rfloor$ d'après la proposition 3.5, et par $\operatorname{card}(A)$. On applique alors la proposition 2.2, et l'on obtient la borne supérieure annoncée.

Cette borne est atteinte : en effet, posons $t = \min\{1 + \lfloor \log_2 m \rfloor, \operatorname{card}(A)\}$ et choisissons t lettres distinctes dans A, disons $a_1, a_2, \ldots, a_t$; si le mot $\xi(a_1 a_2 \cdots a_{t-1})\,a_t$ est préfixe de x, alors $nfs_x(\xi(a_1 a_2 \cdots a_{t-1})) = t$.

■

– Phase d'initialisation.
$carr[0] \leftarrow \{\}$
pour i de 1 à m **faire**
    $j \leftarrow \max(\{0\} \cup \{\, k \mid k \in carr[i-1] \text{ et } x[k] = x[i] \,\})$
    si $j \neq 0$ **alors**
        $carr[i-1] \leftarrow carr[i-1] \setminus \{j\}$
    $carr[i] \leftarrow carr[j] \cup \{j+1\}$
fin pour
– Phase de recherche.
$i \leftarrow 0$
pour j de 1 à n **faire**
    $i' \leftarrow \max(\{0\} \cup \{\, k \mid k \in carr[i] \text{ et } x[k] = y[j] \,\})$
    si $i < m$ et $i' = 0$ et $x[i+1] = y[j]$ **alors**
        $i \leftarrow i+1$
    sinon
        $i \leftarrow i'$
    si $i = m$ **alors**
        Signaler une occurrence du motif x à la position $j - m + 1$.
fin pour

Figure 5: Schéma général des algorithmes du théorème 4.6.

Il s'en déduit donc que :

Corollaire 4.5 *Le délai exact de l'algorithme* MPS *est égal, dans le pire des cas, à* $\min\{1 + \lfloor \log_2 m \rfloor, \operatorname{card}(A)\}$.

Nous nous intéressons maintenant à la classe des algorithmes de $\mathcal{SA}(x)$ qui n'examinent la flèche avant de l'état courant qu'après avoir examiné toutes les flèches arrière (sauf bien entendu si l'état courant est l'état terminal).

Un schéma général des algorithmes de ce type est proposé figure 5 ; par analogie avec le schéma de la figure 2, le tableau *carr* correspond aux seules cibles arrière.

Les algorithmes de cette classe présentent les mêmes complexités globales que les autres algorithmes de $\mathcal{SA}(x)$ (espace $O(m)$, temps d'initialisation $O(m)$ et temps de recherche $O(n)$), mais ils effectuent moins de comparaisons dans le pire des cas :

Théorème 4.6 *N'importe quel algorithme de* $\mathcal{SA}(x)$ *qui n'examine la flèche avant issue de l'état courant qu'après avoir examiné les flèches arrière exécute moins de* $\lfloor (2 - \frac{1}{m})\, n \rfloor$ *comparaisons entre caractères du motif et caractères du texte.*

Preuve. La démonstration est pratiquement identique à celle de la proposition 4.2, mais cette fois, si (p, a, q) est une flèche arrière, le nombre de comparaisons effectuées lors de la boucle $q, \ldots, p$ égale $nfs_x(q \ldots p) - \mathbf{1}\{p \neq x\}$ dans le pire des cas ; on en déduit facilement que pour toute flèche arrière ou nulle (p, a, q), le nombre de comparaisons effectuées lors de la boucle $q, \ldots, p$ est majoré par $(2 - \frac{1}{m})(|p| - |q| + 1)$.

La borne résultante, $\lfloor (2 - \frac{1}{m})\, n \rfloor$, est atteinte lorsque $x = ab^{m-1}$, avec $a, b \in A$ et $a \neq b$, et y est un préfixe d'un mot de l'ensemble x^*.

■

5 Bornes inférieures du problème

Dans cette section, nous donnons des bornes inférieures du problème de la recherche d'un motif par un algorithme séquentiel lorsque l'aphabet est inconnu de l'algorithme et contient au moins deux lettres. Ces bornes impliquent l'optimalité de la phase de recherche des algorithmes du théorème 4.6.

Théorème 5.1 *Dans le pire des cas, n'importe quel algorithme séquentiel de recherche d'un motif effectue au moins* $\lfloor (2 - \frac{1}{m})\, n \rfloor$ *comparaisons.*

Preuve. Considérons le cas où $x = ab^{m-1}$, avec $a, b \in A$ et $a \neq b$.

Soit j, $1 \leq j \leq n$, la position courante de la fenêtre sur le texte, et supposons que l'algorithme vienne juste de découvrir un préfixe propre de longueur non nulle p de x. Puisque l'algorithme ne connaît ni la fin du texte ni l'alphabet, il doit poser les deux questions "$y[j] = a$?" et "$y[j] = b$?" dans le pire des cas, de manière à pouvoir signaler plus tard une occurrence du motif x qui commence à la position j ou à la position $j - |p|$. Si la réponse à une première de ces deux questions est "non" et si celle à la seconde est "oui", l'algorithme découvre à nouveau un préfixe propre de longueur non nulle de x, sauf peut-être dans le cas où $|p| = m - 2$. Il s'en déduit que si $y[j] = a$, l'algorithme effectue 2 comparaisons en chacune des $\min\{m - 1, n - j\}$ positions suivantes dans le pire des cas.

Par conséquent, si $y[j] = a$ pour chaque j tel que $1 \leq j \leq n$ et $j \bmod m = 1$, le nombre de comparaisons exécutées sur le texte dans son entier n'est jamais inférieur à $2n - 1 - \lfloor \frac{n-1}{m} \rfloor$ ($= \lfloor (2 - \frac{1}{m})\, n \rfloor$) dans le pire des cas.

■

Théorème 5.2 *Dans le pire des cas, n'importe quel algorithme séquentiel de recherche d'un motif a un délai au moins égal à* $\min\{1 + \lfloor \log_2 m \rfloor, \mathrm{card}(A)\}$.

Preuve. Comme nous l'avons fait dans la preuve du théorème 4.4, posons $t = \min\{1 + \lfloor \log_2 m \rfloor, \mathrm{card}(A)\}$, choisissons t lettres distinctes a_1, a_2, ..., a_t de A, et considérons le cas où $\xi(a_1 a_2 \cdots a_{t-1})\, a_t$ est préfixe de x.

Si $\xi(a_1 a_2 \cdots a_{t-1})$ est préfixe de y, l'algorithme peut supposer qu'une occurrence du motif x commence à l'une des positions 2^{t-i}, avec $1 \leq i \leq t$. Il effectue donc t comparaisons à la position 2^{t-1} dans le pire des cas.

■

6 Conclusions

Nous avons donné les bornes exactes du nombre de comparaisons pour le problème de la recherche d'un motif dans un texte par un algorithme séquentiel lorsque l'alphabet A est quelconque et contient au moins 2 lettres : $\lfloor (2 - \frac{1}{m})\, n \rfloor$ pour tout le texte, et $\min\{1 + \lfloor \log_2 m \rfloor, \mathrm{card}(A)\}$ en chacun des caractères du texte, où m est la longueur du mot et n celle du texte.

Ces résultats s'étendent facilement au cas particulier où l'algorithme *connaît* l'alphabet. La borne inférieure $\lfloor (2 - \frac{1}{m})\, n \rfloor$ montrée dans le théorème 5.1 reste

valide si $\mathrm{card}(A) \geq 3$. La borne inférieure du délai montrée dans le théorème 5.2 est réduite à $\min\{1+\lfloor \log_2 m \rfloor, \mathrm{card}(A)-1\}$; mais n'importe quel algorithme de $\mathcal{SA}(x)$ peut être modifié de manière à n'effectuer jamais plus de $\mathrm{card}(A)-1$ comparaisons sur chaque caractère du texte. Enfin, si A est un alphabet binaire, le délai égale 1 ; la recherche du motif s'effectue en n comparaisons et en temps réel, comme mentionné dans [10].

D. Breslauer, L. Colussi et L. Toniolo ont récemment étudié le problème du *string-prefix-matching* qui consiste à calculer pour chacun des préfixes p du texte y le plus long préfixe du motif x qui soit aussi suffixe de p. Clairement, tout algorithme séquentiel de recherche d'un motif permet de résoudre ce problème. Les auteurs ont montré une borne optimale de $\lfloor (2-\frac{1}{m})\, n \rfloor$ comparaisons lorsque l'alphabet est quelconque. En adaptant simplement la preuve du théorème 5.2, il vient que tout algorithme solution de ce problème présente un pire des cas de $\min\{1+\lfloor \log_2 m \rfloor, \mathrm{card}(A)\}$ comparaisons sur chaque caractère du texte ; cette borne est donc, dans ce cas encore, optimale.

Références

[1] A.V. Aho, J.E. Hopcroft, et J.D. Ullman, *The design and analysis of computer algorithm*, Addison-Wesley, Reading, MA. (1974).

[2] D. Beauquier, J. Berstel, et P. Chrétienne, *Eléments d'algorithmique*, Masson, Paris (1992).

[3] D. Breslauer, L. Colussi, et L. Toniolo, Tight comparison bounds for the string prefix-matching problem, *Information Processing Letters* **47** (1993) 51–57.

[4] R. Cole et R. Hariharan, Tighter bounds on the exact complexity of string matching, In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science* (1992) 600–609.

[5] M. Crochemore, Off-line serial exact string searching, Rapport de recherche 92.1, Institut Gaspard Monge, Université de Marne-la-Vallée (1992).

[6] J.P. Duval, A remark on the Knuth-Morris-Pratt string searching algorithm, Manuscrit (1981).

[7] Z. Galil et R. Giancarlo, On the exact complexity of string matching : lower bounds, *SIAM Journal on Computing* **20** (6) (1991) 1008–1020.

[8] Z. Galil et R. Giancarlo, On the exact complexity of string matching : upper bounds, *SIAM Journal on Computing* **21** (3) (1992) 407–437.

[9] C. Hancart, On Simon's string searching algorithm, *Information Processing Letters* **47** (1993) 95–99.

[10] D.E. Knuth, J.H. Morris, et V.R. Pratt, Fast pattern matching in strings, *SIAM Journal on Computing* **6** (1) (1977) 323–350.

[11] D. Perrin, Finite automata,. In *Handbook of Theoretical Computer Science*, volume B, 1–57, J. van Leeuwen, Amsterdam (1990).

[12] I. Simon, Communication à M. Crochemore (1989).

[13] I. Simon, String matching algorithms and automata, In R. Baeza-Yates et N. Ziviani, éditeurs, *First South-American Workshop on String Processing*, 151–157. Belo Horizonte, Brazil (1993).

Christophe Hancart
Institut Gaspard Monge
Université de Marne-la-Vallée
2, rue de la Butte Verte
F-93166 Noisy-le-Grand Cedex
France