RAIRO. INFORMATIQUE THÉORIQUE ET APPLICATIONS

JEAN-JACQUES HEBRARD
MAXIME CROCHEMORE

Calcul de la distance par les sous-mots

RAIRO. Informatique théorique et applications, tome 20, n° 4 (1986), p. 441-456

<http://www.numdam.org/item?id=ITA_1986__20_4_441_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 20, n° 4, 1986, p. 441 à 456)

CALCUL DE LA DISTANCE PAR LES SOUS-MOTS (*)

par Jean-Jacques HEBRARD ([1]) et Maxime CROCHEMORE ([2])

Communiqué par J. BERSTEL

Résumé. – *Cet article contient deux méthodes de calcul de la longueur du plus court sous-mot (au sens de sous-suite) permettant de distinguer deux mots différents u et v. L'utilisation d'automates et des structures de données concernant les questions d'« union et recherche » conduit à un algorithme quasi linéaire en la longueur de uv.*

Abstract. – *This paper gives two methods to compute the shortest subsequence which distinguishes two different words u and v. The use of automata together with data structures for "Union-Find" questions leads to an algorithm almost linear in the length of uv.*

Définir des critères de comparaison entre chaînes de caractères (ou mots) est un problème qui apparaît fréquemment, dans des domaines aussi divers que la biochimie, la reconnaissance de la parole, l'informatique (correction de mots mal orthographiés, recherche d'informations non entièrement spécifiées), la reconnaissance de contours, etc. [SK 83].

Parmi les critères les plus couramment utilisés on trouve la distance de Hamming et la distance d'édition [Mo 70, Se 74, WF 74, MP 80]. Ces deux distances sont fondamentalement différentes : la première n'est qu'une stricte comparaison, caractère à caractère, des deux mots, alors que la seconde prend en compte de façon essentielle, les similitudes entre leurs sous-mots (un sous-mot d'un mot est obtenu en supprimant dans ce dernier un certain nombre de caractères).

Il est souvent possible de ramener la comparaison de deux mots u et v à celle de leurs ensembles respectifs de sous-mots. Les deux indices de proximité les plus simples sont alors : la longueur maximale $L(u, v)$ d'un sous-mot

(*) Reçu juillet 1985, révisé décembre 1985.
([1]) Laboratoire d'Informatique, Univ. Caen, 14032 Caen Cedex.
([2]) L.I.T.P.-Rouen et C.S.P., Univ. Paris-Nord, avenue J.-B.-Clément, 93400 Villetaneuse.

Informatique théorique et Applications/Theoretical Informatics and Applications
0296-1598 86/04 441 16/$3.60/

commun, et la « distance par les sous-mots » ($d(u, v)$) définie à l'aide de la longueur minimale des mots qui distinguent u et v — c'est-à-dire, des mots qui ne sont pas des sous-mots communs à u et v.

Le calcul de $L(u, v)$ est traité dans de nombreuses études; citons entre autres celle de Hirschberg [Hi 77] et Nakatsu, Kambayashi et Yajima [NKY 82] dont les méthodes sont de complexité maximale quadratique. L'algorithme de Hunt et Szymanski [HS 77], bien que de complexité maximale $O((|u|+|v|)^2 \log(|u|+|v|))$, est d'une grande efficacité dans certaines situations pratiques; il est d'ailleurs utilisé dans la mise en œuvre de la commande DIFF de UNIX.

Les aspects théoriques de l'étude de la « distance par les sous-mots » sont développés par I. Simon dans [Lo 82]. Il en déduit un algorithme linéaire de calcul de $d(u, v)$ qui est malheureusement très complexe et malaisé à mettre en œuvre [Si 84]. Une analyse directe du problème est réalisée dans [He 84]; elle conduit à un algorithme linéaire pour un alphabet à deux lettres.

Nous adoptons une autre approche, fondée essentiellement sur le fait que l'on peut construire en temps linéaire, pour un mot donné u, un automate $\mathscr{S}(u)$ qui reconnaît l'ensemble des sous-mots de u. Le calcul de $L(u, v)$ se réduit alors à la recherche d'un chemin de longueur maximale dans l'automate produit $\mathscr{S}(u) \times \mathscr{S}(v)$; cet algorithme est de complexité quadratique [He 84]. Nous présentons ici les deux algorithmes que l'on obtient en utilisant l'automate des sous-mots pour le calcul de $d(u, v)$, Le premier est l'application à l'automate union $\mathscr{S}(u) \cup \mathscr{S}(v)$ d'une méthode de partitionnement; sa complexité en temps est $O((|u|+|v|)\,\mathrm{Log}(|u|+|v|))$. Le second consiste à ramener le calcul à un problème d'équivalence d'automates; la mise en œuvre est immédiate et très efficace : sa complexité est quasi linéaire en temps et linéaire en espace.

1. AUTOMATES ET SOUS-MOTS

1.1. Définitions générales

Soit $(A, \leqq)$ un alphabet fini totalement ordonné. A^* est l'ensemble des mots formés sur A. On note ε le mot vide et $|w|$ la longueur d'un mot w.

On considère l'ordre généalogique sur A^*, noté $\leqq$ également, et défini de la façon suivante : pour tout $u, v \in A^*$, $u \leqq v$ si et seulement si :

ou bien $|u| < |v|$;

ou bien $|u| = |v|$, $u = \mathrm{ras}$, $v = rbt$, avec $r, s, t \in A^*$, $a, b \in A$ et $a \leqq b$;

ou bien $u = v$.

Si $u \leqq v$, alors soit v est strictement plus long que u, soit u et v sont de même longueur et v est plus grand que u pour l'ordre lexicographique.

Soient $u = u_1 \ldots u_m$ $(u_i \in A)$ et $v = v_1 \ldots v_n$ $(v_j \in A)$ deux mots de A^*. On dit que v est un sous-mot de u s'il existe une application $f : \{1, \ldots, n\} \to \{1, \ldots, m\}$ strictement croissante telle que $v_1 \ldots v_n = u_{f(1)} \ldots u_{f(n)}$.

Exemple :

$$A = \{a, b, c\}, \qquad u = abcabc, \qquad v = bcc.$$

REMARQUE : Un mot peut être sous-mot d'un autre de plusieurs manières. Ainsi *abc* est un sous-mot de *abcabc* de quatre façons différentes.

Si v est un sous-mot de u, on dit aussi que v divise u et on note $v \mid u$. Si v ne divise pas u, on note $v + u$.

Un automate fini déterministe complet est un quadruplet $\mathscr{A} = (Q, i, T, t)$, où Q est l'ensemble fini des états, i l'état initial, T l'ensemble des états terminaux et t l'application de $Q \times A$ dans Q qui définit les transitions de $\mathscr{A}$; t se prolonge en une application de $Q \times A^*$ dans Q. On emploie la notation abrégée : $\forall q \in Q, \forall w \in A^*, q.w = t(q, w)$.

Le language reconnu par l'automate $\mathscr{A}$ est l'ensemble $|\mathscr{A}| = \{w \in A^* / i.w \in T\}$.

1.2. Distance par les sous-mots

Étant donné un mot u et un entier l, on note $S(u, l)$ l'ensemble des sous-mots w de u tels que $|w| \leqq l$.

$$S(u, l) = \{w \in A^* / w \mid u \text{ et } |w| \leqq l\}.$$

On appelle distance par les sous-mots de deux mots u et v, l'élément de $N \cup \{\infty\}$, noté $d(u, v)$, et défini par :

$$d(u, v) = \max\{l \in N / S(u, l) = S(v, l)\} \qquad \text{si} \quad u \neq v,$$

$$d(u, v) = \infty \quad \textit{sinon.}$$

Évidemment, $d(u, v)$ n'est pas une distance au sens usuel, mais $\delta(u, v) = 2^{-d(u, v)}$ est une distance ultramétrique. On trouve une étude détaillée de $d(u, v)$ dans [Lo 82].

On dit qu'un mot w distingue deux mots u et v, s'il divise l'un et ne divise pas l'autre. Si w est un mot de longueur minimale qui distingue u et v alors $d(u, v) = |w| - 1$.

Exemple :

$$u = ababa, \qquad v = aabba, \qquad S(u, 2) = S(v, 2),$$
$$bab \mid u, \qquad bab + v, \qquad d(u, v) = 2.$$

1.3. Automate des sous-mots

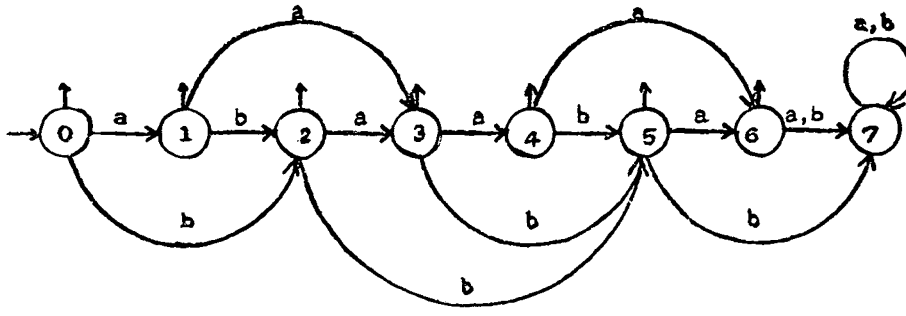
On peut construire, en temps linéaire, l'automate minimal qui reconnaît l'ensemble des sous-mots d'un mot donné.

Étant donné un mot $u = u_1 \ldots u_m$ $(u_i \in A)$, on note $\mathscr{S}(u)$ l'automate fini déterministe défini par :

$$\mathscr{S}(u) = \{\{0, \ldots, m+1\}, 0, \{0, \ldots, m\}, t\},$$
$$\forall a \in A, \quad t(m, a) = t(m+1, a) = m+1,$$
$$\forall a \in A, \quad \forall i \in \{0, \ldots, m-1\}, \quad E_a(i) = \{k \in \{i+1, \ldots, m\} / u_k = a\},$$
$$t(i, a) = \begin{cases} \min E_A(i) & \text{si} \quad E_a(i) \neq \varnothing \\ m+1, & \text{sinon.} \end{cases}$$

Exemple :

$$A = \{a, b\}, \qquad u = abaaba, \qquad m = 6.$$



On a les propriétés suivantes :

(1) $\forall i, \quad j \in \{0, \ldots, m+1\}, \quad \forall w \in A^*, \qquad i \leqq j \quad \Rightarrow \quad t(i, w) \leqq t(j, w).$

(2) $\qquad \forall i, \quad j \in \{0, \ldots, m+1\}, \qquad i < j \quad \Rightarrow \quad t(i, u_j) \leqq j.$

PROPOSITION 1 : *Pour tout* $u = u_1 \ldots u_m (u_i \in A)$, $\mathscr{S}(u)$ *reconnaît l'ensemble des sous-mots de u.*

Démonstration : Pour tout mot $w = w_1 \ldots w_p$ $(w_i \in A)$ et tout $i \in \{1, \ldots, p\}$, soit $g(i) = t(0, w_1 \ldots w_i)$. Il suffit de montrer : $g(p) \leqq m \Leftrightarrow w \mid u$. Si $g(p) \leqq m$,

alors g est une application strictement croissante de $\{1, \ldots, m\}$ qui fait de w un sous-mot de u.

Si $w \mid u$, alors $w = u_{f(1)} \ldots u_{f(p)}$, où f est une application strictement croissante de $\{1, \ldots, p\}$ dans $\{1, \ldots, m\}$. Par définition de $\mathscr{S}(u)$, $g(1) \leqq f(1)$ et pour tout i de $\{2, \ldots, p\}$,

$$g(i-1) \leqq f(i-1) \Rightarrow g(i-1) < f(i)$$
$$\Rightarrow t(g(i-1), u_{f(i)}) \leqq f(i) \Rightarrow g(i) \leqq f(i)$$

et donc $g(p) \leqq f(p) \leqq m$. ■

$\mathscr{S}(u)$ est appelé l'automate des sous-mots de u. Sa construction est réalisée par l'algorithme suivant, en un simple parcours de u de droite à gauche.

ALGORITHME 2 :

Entrée : $u = u_1 \ldots u_m (u_i \in A)$
Sortie : t, fonction de transition de $\mathscr{S}(u)$

début
pour chaque lettre a de A faire début $t(m+1, a) \leftarrow m+1$; $t(m, a) \leftarrow m+1$ fin;
pour $i \leftarrow m-1$ à 0 pas -1 faire
début
$t(i, u_{i+1}) \leftarrow i+1$;
pour chaque lettre a de $A \setminus \{u_{i+1}\}$ faire $t(i, a) \leftarrow t(i+1, a)$;
fin
fin.

PROPOSITION 3 : *Pour tout mot u, l'algorithme 2 calcule la fonction de transition de $\mathscr{S}(u)$; sa complexité est $O(|A|\,|u|)$.*

Démonstration : Ceci est une conséquence de la définition de $\mathscr{S}(u)$. ■

2. PARTITIONNEMENT

Le calcul de $d(u, v)$ se ramène à la construction d'une suite d'équivalences $R(k)$ $(k \in N)$ sur l'ensemble des états de l'automate union $\mathscr{S}(u) \cup \mathscr{S}(v)$. Pour tout k, $R(k+1)$ est plus fine que $R(k)$. On montre que $d(u, v) = D$ si et seulement si D est le plus petit entier tel que les états initiaux de $S(u)$ et $\mathscr{S}(v)$ ne sont pas équivalents pour $R(D+1)$. La construction de la suite R est réalisée en adaptant l'algorithme de partitionnement de Hopcroft [AHU 74], le calcul de $d(u, v)$ est alors effectué en temps $O(|A|(|u|+|v|)\log(|u|+|v|))$.

Considérons les mots $u = u_1 \ldots u_m$ $(u_i \in A)$, $v = v_1 \ldots v_n$ $(v_j \in A)$ et les automates $\mathscr{S}(u) = (\{0, \ldots, m+1\}, 0, \{0, \ldots, m\}, t_u)$ et $\mathscr{S}(v) = (\{\bar{0}, \ldots, \overline{n+1}\}, \bar{0},$

$\{\bar{0}, \ldots, \bar{n}\}, t_v)$. Soit $\mathscr{S}(u) \cup \mathscr{S}(v) = (Q, I, T, t)$ l'automate union. On convient de confondre $m+1$ et $\overline{n+1}$ en un seul état noté « Puits ».

$$Q = \{0, \ldots, m\} \cup \{\bar{0}, \ldots, \bar{n}\} \cup \{\text{Puits}\},$$
$$I = \{0, \bar{0}\}, \qquad T = Q \setminus \{\text{Puits}\}.$$

Cet automate n'est pas déterministe.

Pour tout entier k, $R(k)$ est la relation d'équivalence définie sur Q par :

$$\forall p, \quad q \in Q, \qquad p \equiv q\,[R(k)]$$
$$\Leftrightarrow \quad (\forall w \in A^*, \ |w| \leqq k \Rightarrow (p.w \in T \Leftrightarrow q.w \in T)).$$

On note $P(k)$ la partition associée à $R(k)$. Ainsi $P(0) = \{\{\text{Puits}\}, Q \setminus \{\text{Puits}\}\}$.

La proposition suivante, dont la preuve est une simple transcription des définitions de d et de la suite d'équivalences R, fournit un moyen de calcul de d. Elle exprime uniquement que la distance $d(u, v)$ entre deux mots u et v est le plus grand entier D pour lequel les états initiaux de $\mathscr{S}(u)$ et $\mathscr{S}(v)$ sont $R(D)$-équivalents.

PROPOSITION 4 : $\forall D \in N, \quad d(u, v) = D \Leftrightarrow (\forall k \leqq D, \ 0 \equiv \bar{0}\,[R(k)])$ *et* $0 \not\equiv \bar{0}\,[R(D+1)]$.

Démonstration :
$\forall k \in N, \ 0 \equiv \bar{0}\,[R(k)] \Leftrightarrow (\forall w \in A^*, \ |w| \leqq k \Rightarrow (w|u \Leftrightarrow w|v))$. ■

REMARQUES : (1) Pour tout k, $R(k+1)$ est plus fine que $R(k)$ (on note $R(k) \geqq R(k+1)$).

(2) Pour tout k, $R(k) \neq R(k+1) \Leftrightarrow (\exists p, q \in Q, \ \exists a \in A, \ p \equiv q\,[R(k)]$ *et* $p.a \not\equiv q.a\,[R(k)])$.

(3) Il existe un entier K tel que $R(0) > \ldots > R(K)$ et $\forall k > K$, $R(k) = R(K)$.

$$(\text{où } R(k) > R(k') \Leftrightarrow (R(k) \geqq R(k') \text{ et } R(k) \neq R(k')))$$

Q étant fini, l'ensemble $E = \{k \in N / R(k) = R(k) = R(k+1)\}$ est non vide, et $K = \min E$.

(4) $R(K)$ est l'équivalence la plus grossière qui soit à la fois plus fine que $R(0)$ et compatible avec t :

$$\forall p, \quad q \in Q, \quad \forall a \in A, \qquad p \equiv q\,[R(K)] \quad \Rightarrow \quad p.a \equiv q.a\,[R(K)].$$

Le calcul de $R(K)$ est un problème de partitionnement qui peut être résolu en utilisant l'algorithme de Hopcroft [AHU 74]. Cependant celui-ci ne permet

pas de construire effectivement la suite R. Aussi emploierons-nous une méthode dérivée, définie dans [CC 82] pour le partitionnement d'un graphe.

NOTATIONS :

– $\forall q \in Q, \forall a \in A, qa^{-1} = \{p \in Q / p.a = q\}$.

– $\forall B \subset Q, \forall a \in A, Ba^{-1} = \bigcup_{q \in B} qa^{-1}$.

– $P = \{B_1, \ldots, B_h\}$ étant une partition de Q, on note $[B_1, \ldots, B_h]$ l'équivalence associée.

– L'intersection de deux équivalences est une opération associative; soient R_1 et R_2 deux équivalences sur Q, on a :

$$\forall p, \quad q \in Q, \quad p \equiv q\,[R_1 \cap R_2] \Leftrightarrow (p \equiv q\,[R_1] \text{ et } p \equiv q\,[R_2]).$$

Pour toute partition P de Q et toute partie T de $\mathscr{P}(Q)$, on note $P \Lambda T$ la partition associée à l'équivalence R définie par :

$$\forall p, \quad q \in Q, \quad p \equiv q\,[R] \Leftrightarrow (\forall B \in P, \quad \forall C \in T \cup \{Q\}, p \in B \cap C \Leftrightarrow q \in B \cap C).$$

Une première construction de la suite R (k) $(k \in N)$ est donnée par la :

PROPOSITION 5 : *Pour tout* k, $R(k+1) = \bigcap_{\substack{B \in P(k) \\ a \in A}} [Ba^{-1}, Q \setminus Ba^{-1}]$.

Démonstration : Soit

$$E = \bigcap_{\substack{B \in P(k) \\ a \in A}} [Ba^{-1}, Q \setminus Ba^{-1}].$$

$$\forall p, \quad q \in Q, \quad p \equiv q\,[E] \Leftrightarrow \forall a \in A, \quad p.a \equiv q.a\,[R(k)] \Leftrightarrow p \equiv q\,[R(k+1)]. \quad \blacksquare$$

On améliore cette méthode en distinguant les petites et les grandes classes. Soient

$$k > 0, \quad P(k) = \{B_1, \ldots, B_h\} \quad \text{et} \quad P(k-1) = \{B'_1, \ldots, B'_{h'}\};$$

toute classe B'_i suivant $R(k-1)$ est union de classes suivant $R(k)$. On définit une application g qui permet d'associer à toute classe B'_i de $P(k-1)$ une classe de $P(k)$ qu'elle contient et qui est de cardinal maximal,

$$g : P(k-1) \to P(k), \quad \forall i \in \{1, \ldots, h'\}, \quad g(B'_i) \subset B'_i$$

et

$$\forall B \in P(k), \quad B \subset B_i' \quad \Rightarrow \quad |B| \leqq |g(B_i')|.$$

Les classes qui sont dans l'image de g sont appelées les grandes classes, les autres les petites classes. On note $Pt(k)$ l'ensemble des petites classes de $P(k)$. On a $Pt(0) = \{\{\text{Puits}\}\}$.

La proposition suivante montre qu'il n'est pas nécessaire d'utiliser toutes les classes de $R(k)$ pour calculer $R(k+1)$, les petites classes suffisent.

PROPOSITION 6 : *Pour tout k,*

$$R(k+1) = R(k) \cap \Big(\bigcap_{\substack{B \in Pt(k) \\ a \in A}} [Ba^{-1}, Q \setminus Ba^{-1}] \Big).$$

Démonstration : $E' = R(k) \cap \Big(\bigcap_{\substack{B \in Pt(k) \\ a \in A}} [Ba^{-1}, Q \setminus Ba^{-1}] \Big)$.

Si $k = 0$, $Pt(k) = \{\{\text{Puits}\}\}$; $\forall p, q \in Q$, $p \equiv q\,[E'] \Leftrightarrow \forall a \in A$, $p.a \equiv q.a\,[R(0)]$.

Si $k > 0$, $R(k) = \bigcap_{\substack{C \in P(k-1) \\ a \in A}} [Ca^{-1}, Q \setminus Ca^{-1}]$ (prop. 5) :

$$P(k-1) \wedge Pt(k) = P(k),$$

donc

$$E' = \bigcap_{\substack{B \in P(k) \\ a \in A}} [Ba^{-1}, Q \setminus Ba^{-1}] = R(k+1). \quad \blacksquare$$

REMARQUE : Pour tout k, on a : $Pt(k) = \varnothing \Leftrightarrow R(k) = R(k-1)$.

L'algorithme de calcul de $d(u, v)$ est décrit par le schéma suivant :

ALGORITHME 7. — Fonction $d(u, v)$

début
$k \leftarrow 0$; $R(k) \leftarrow [\{\text{Puits}\}, Q \setminus \{\text{Puits}\}]$; $Pt(k) \leftarrow \{\{\text{Puits}\}\}$;
tant que $0 \equiv \bar{0}\,[R(k)]$ et $Pt(k) \neq \varnothing$ faire
début $R(k+1) \leftarrow R(k) \cap \Big(\bigcap_{\substack{B \in Pt(k) \\ a \in A}} [Ba^{-1}, Q \setminus Ba^{-1}] \Big)$;
$k \leftarrow k+1$
fin;
si $0 \not\equiv \bar{0}\,[R(k)]$ alors retour $(k-1)$
sinon retour (∞)
fin

PROPOSITION 8 : *L'algorithme 7 effectue le calcul de $d(u, v)$ en temps $O(|A|(|u|+|v|)\log(|u|+|v|))$.*

Démonstration : On peut mettre en œuvre la construction de la suite $R(k)$ ($k \in N$) de façon que le temps d'exécution soit proportionnel au nombre d'opérations $q a^{-1}$ effectuées ($q \in Q$, $a \in A$) [CC 82]. Soit α ce nombre pour q et a fixés. L'opération $q a^{-1}$ est effectuée à chaque fois que l'état q se trouve dans une petite classe. Ceci se produit au plus $\log_2 |Q|$ fois — en effet, pour toute petite classe B de $R(k)$ contenue dans une classe B' de $R(k-1)$ on a : $|B| \leqq |B'|/2$. Par conséquent $\alpha \leqq \log_2 |Q|$, d'où le résultat.■

Exemple :

$$A = \{a, b, c\}, \qquad u = cabacb, \qquad v = bacabc.$$

$\mathscr{S}(u)$: ⓪ c ① a ② b ③ a ④ c ⑤ b ⑥

Puits

$\mathscr{S}(v)$: ⓪ b ① a ② c ③ a ④ b ⑤ c ⑥

$$P(0) = \{\{0, 1, 2, 3, 4, 5, 6, \bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}, \bar{6}\}, \{\text{Puits}\}\},$$

$$Pt(0) = \{\{\text{Puits}\}\},$$

$$\{\text{Puits}\}\ a^{-1} = \{4, 5, 6, \bar{4}, \bar{5}, \bar{6}, \text{Puits}\}, \qquad \{\text{Puits}\}\, b^{-1} = \{6, \bar{5}, \bar{6}, \text{Puits}\},$$

$$\{\text{Puits}\}\, c^{-1} = \{5, 6, \bar{6}, \text{Puits}\}.$$

$$P(1) = \{\{0, 1, 2, 3, \bar{0}, \bar{1}, \bar{2}, \bar{3}\}, \{4, \bar{4}\}, \{5\}, \{\bar{5}\}, \{6, \bar{6}\}, \{\text{Puits}\}\}$$

$$Pt(1) = \{\{4, \bar{4}\}, \{5\}, \{\bar{5}\}, \{6, \bar{6}\}\}$$

$$\{4, \bar{4}\}\, a^{-1} = \{2, 3, \bar{2}, \bar{3}\}, \qquad \{4, \bar{4}\}\, b^{-1} = \varnothing, \qquad \{4, \bar{4}\}\, c^{-1} = \varnothing,$$

$$\{5\}\, a^{-1} = \varnothing, \qquad \{5\}\, b^{-1} =, \qquad \{5\}\, c^{-1} = \{1, 2, 3, 4\},$$

$$\{\bar{5}\}\, a^{-1} = \varnothing, \qquad \{\bar{5}\}\, b^{-1} = \{\bar{1}, \bar{2}, \bar{3}, \bar{4}\}, \qquad \{\bar{5}\}\, c^{-1} = \varnothing,$$

$$\{6, \bar{6}\}\, a^{-1} = \varnothing, \qquad \{6, \bar{6}\}\, b^{-1} = \{3, 4, 5\}, \qquad \{6, \bar{6}\}\, c^{-1} = \{\bar{3}, \bar{4}, \bar{5}\}.$$

$$P(2) = \{\{0, \bar{0}\}, \{1\}, \{\bar{1}\}, \{2\}, \{\bar{2}\}, \{3\}, \{\bar{3}\}, \{4\}, \{4\}, \{5\}, \{\bar{5}\}, \{6, \bar{6}\}, \{\text{Puits}\}\}$$

$$Pt(2) = \{\{1\}, \{\bar{1}\}, \{2\}, \{\bar{2}\}, \{3\}, \{\bar{3}\}, \{4\}\}$$

$$\{1\}\, a^{-1} = \varnothing, \qquad \{1\}\, b^{-1} =, \qquad \{1\}\, c^{-1} = \{0\}$$

$$\{\bar{1}\}\, a^{-1} = \varnothing. \qquad (\bar{1})\, b^{-1} = \{\bar{0}\}, \qquad \{1\}\, c^{-1} = \varnothing$$

$$\{2\}\, a^{-1} = \{0, 1\}, \qquad \{2\}\, b^{-1} = \varnothing, \qquad \{2\}\, c^{-1} = \varnothing$$

$$\{\bar{2}\}\, a^{-1} = \{\bar{0}, \bar{1}\}, \qquad \{\bar{2}\}\, b^{-1} = \varnothing, \qquad \{\bar{2}\}\, c^{-1} = \varnothing,$$

$$\{3\}\, a^{-1} = \varnothing, \qquad \{3\}\, b^{-1} = \{0, 1, 2\}, \qquad \{3\}\, c^{-1} = \varnothing$$

$$\{\bar{3}\}\, a^{-1} = \varnothing, \qquad \{\bar{3}\}\, b^{-1} = \varnothing, \qquad \{\bar{3}\}\, c^{-1} = \{\bar{0}, \bar{1}, \bar{2}\}$$

$$\{4\}\, a^{-1} = \{2, 3\}, \qquad \{4\}\, b^{-1} = \varnothing, \qquad \{4\}\, c^{-1} = \varnothing.$$

$P(3)=\{\{0\}, \{\bar{0}\}, \{1\}, \{\bar{1}\}, \{2\}, \{\bar{2}\}, \{3\}, \{\bar{3}\},$
$\{4\}, \{\bar{4}\}, \{5\}, \{\bar{5}\}, \{\bar{6}\}, \{\text{Puits}\}$

$0 \not\equiv \bar{0}\,[R(3)]$ et $d(u, v)=2$.

3. UN ALGORITHME QUASI LINEAIRE

Le calcul de la distance par les sous-mots est un problème voisin de celui de l'équivalence de deux automates [AHU 74]. En effet, le test d'équivalence de $\mathscr{S}(u)$ et $\mathscr{S}(v)$, et le calcul de $d(u, v)$, conduisent tous les deux à l'étude de la différence symétrique Δ de $|\mathscr{S}(u)|$ et $|\mathscr{S}(v)|$. Dans un cas on teste si Δ est vide, dans l'autre on calcule la longueur minimale des mots de Δ. En fait, la méthode que nous présentons est plus précise encore : il s'agit de déterminer le mot h de Δ le plus petit pour l'ordre généalogique; on a alors $d(u, v)=|h|-1$. Ceci est obtenu en générant une suite de mots finie, croissante et contenant nécessairement le mot h s'il existe. Naturellement, le calcul est d'autant plus rapide que la suite comporte moins d'éléments. Nous considérons successivement deux suites. La première, qui donne un algorithme quadratique, correspond au parcours en largeur du graphe de l'automate produit $\mathscr{S}(u) \times (v)$. La construction de la seconde, sous-suite de la précédente, se ramène à un problème de type « UNION-FIND » pour lequel il existe une solution simple et de complexité optimale [AHU 74]; elle permet d'effectuer le calcul de $d(u, v)$ en temps quasi linéaire.

Considérons les deux mots $u=u_1 \ldots u_m$ $(u_i \in A)$, $v=v_1 \ldots v_n$ $(v_i \in A)$ et les automates

$$\mathscr{S}(u)=(\{0, \ldots, m+1\}, 0, \{0, \ldots m\}, t_u),$$
$$\mathscr{S}(v)=(\{\bar{0}, \ldots, \overline{n+1}\}, \bar{0}, \{\bar{0}, \ldots, \bar{n}\}, t_v).$$

Soient

$$Z=\{w \in A^* / \forall x \in A^*, (0.w, \bar{0}.w)=(0.x, \bar{0}.x) \Rightarrow w \leqq x\},$$

et

$$l=\text{card}\,(Z)\,(\leqq (m+2)(n+2)).$$

Soit $z(i)\,(1 \leqq i \leqq l)$ la suite obtenue en ordonnant, de façon croissante pour l'ordre généalogique les éléments de Z.

On remarque que Z est clos par préfixe :

$$\forall x, \quad y \in A^*, \qquad xy \in Z \quad \Rightarrow \quad x \in Z.$$

Il est alors facile de construire z à l'aide d'une file f. La construction fait appel aux opérations suivantes :

INITIALISER (f) : procédure qui vide f.
AJOUTER (w, f) : procédure qui ajoute l'élément w à la fin de f.
TÊTE (f) : fonction dont le résultat est le premier élément de f.
SUPRIMER (f) : procédure qui supprime le premier élément de f.
MARQUER (p, q) : procédure qui permet de marquer un élément (p, q) de $\{0, \ldots, m+1\} \times \{\bar{0}, \ldots, \overline{n+1}\}$.

Construction de z

```
début
    i ← 1; z(i) ← ε; INITIALISER (f); AJOUTER (ε, f);
    tant que f ≠ file vide faire
        début
            w ← TÊTE(f); SUPPRIMER (f);
            pour a variant de la première à la dernière lettre de A faire
                si (0.wa, 0̄.wa) n'est pas marqué alors
                    début i ← i+1; z(i) ← wa; AJOUTER (wa, f);
                          MARQUER (0.wa, 0̄.wa)
                    fin
        fin {tant que}
fin {construction}
```

La suite z permet de calculer $d(u, v)$. En effet, si $u \neq v$, il est clair que Z contient le mot le plus petit pour l'ordre généalogique qui distingue u et v.

Pratiquement on utilise la suite t d'éléments de $\{0, \ldots, m+1\} \times \{\bar{0}, \ldots, \overline{n+1}\}$ définie par $t(i) = (0.z(i), \bar{0}.z(i))$ $(1 \leqq i \leqq 1)$. Cette suite est celle que l'on obtiendrait en effectuant un parcours en largeur du graphe de l'automate produit $\mathscr{S}(u) \times \mathscr{S}(v)$, les sommets adjacents d'un sommet s donné étant examinés dans l'ordre des étiquettes des flèches issues de s.

Soit $D = \{0, \ldots, m\} \times \{\overline{n+1}\} \cup \{m+1\} \times \{\bar{0}, \ldots, \bar{n}\}$. $z(i)$ distingue u et v si et seulement si $t(i) \in D$. L'algorithme 9 calcule $d(u, v)$, il utilise une file F d'éléments de la forme $(p, \bar{q}, k)$ où $(p, \bar{q}) = t(i)$, $k = |z(i)|$ $(1 \leqq i \leqq 1)$.

ALGORITHME 9 : Fonction $d(u, v)$.

```
Début
    INITIALISER (F); AJOUTER ((0, 0̄, 0), F);
    tant que F ≠ file vide faire
        début
            (p, q̄, k) ← TÊTE(F); SUPRIMER (F);
            pour a variant de la première à la dernière lettre de A faire
```

```
                si (p.a, q̄.a) n'est pas marqué alors
                    début
                        si (p.a, q̄.a)∈D alors retour (k);
                        AJOUTER ((p.a, q̄.a, k+1), F);
                        MARQUER (p.a, q̄.a)
                    fin {si}
            fin; {tant que}
        retour (∞)
    fin {fonction}
```

PROPOSITION 10 : *La complexité maximale de l'algorithme* 9 *est* $O(|A||u||v|)$.

Démonstration : Le test $(p.a, \bar{q}.a) \in D$ et les opérations AJOUTER et MARQUER se font en temps constant. Le temps d'exécution est donc proportionnel au nombre d'éléments placés dans file; celui-ci est inférieur à $(|u|+2)(|v|+2)$. ■

Améliorer l'algorithme 9 : $d(u, v)$ est plus rapidement calculée en utilisant une sous-suite de z.

Soient $z'(i)$ $(1 \leqq i \leqq l')$ une suite de mots et $G(i)=(S, A(i))$ $(1 \leqq i \leqq l')$ une suite de graphes non orientés, définies de la façon suivante : $S=\{0, \ldots, m+1\} \cup \{\bar{0}, \ldots, \overline{n+1}\}$ est l'ensemble des sommets de $G(i)$, $A(i)$ l'ensemble des arêtes; $z'(1)=\varepsilon$ et $A(1)=\varnothing$; supposons connus les éléments

$$z'(1), \ldots, z'(i), \qquad A(1), \ldots A(i),$$

et soit

$$E(i)=\{x \in A^*/z'(i) \leqq x \qquad \text{et} \qquad C_i(0.x) \neq C_i(\bar{0}.x)\}$$

où $C_i(p)$ $(p \in S)$ est la composante connexe de p dans $G(i)$; si $E(i)=\varnothing$ alors $l'=i$, sinon

$$z'(i+1)=\min E(i)$$

et

$$A(i+1)=A(i) \cup \{\{0.z'(i+1), \bar{0}.z'(i+1)\}\}.$$

L'ensemble $Z'=\{z'(i)/1 \leqq i \leqq l'\}$ est clos par préfixe :

$$\forall x, \quad y \in A^*, \qquad xy \in Z' \quad \Rightarrow \quad x \in Z'.$$

Le schéma suivant décrit la construction de la suite z'. COMPOSANTE (p) $(p \in S)$ désigne la composante connexe de p dans le graphe courant et UNION (p, q) $(p, q \in S)$ est une procédure qui effectue l'ajout de l'arête $\{p, q\}$; f est une file de mots.

Construction de z'

```
début
    i ← 1; z'(i) ← ε; INITIALISER (f); AJOUTER (ε, f);
    tant que f ≠ file vide faire
        début
            w ← TÊTE(f); SUPPRIMER (f);
            pour a variant de la première à la dernière lettre de A faire
                si COMPOSANTE (0.wa) ≠ COMPOSANTE (0̄.wa) alors
                    début i ← i+1; z'(i) ← wa; AJOUTER (wa, f);
                          UNION (0.wa, 0̄.wa)
                    fin
        fin {tant que}
fin (construction}
```

La proposition suivante montre que la suite z' permet de calculer $d(u, v)$.

PROPOSITION 11 : *Si $u \neq v$ et si h est le plus petit mot (pour l'ordre généalogique) qui distingue u et v, alors il existe $i \in \{1, \ldots, l'\}$ tel que $z'(i) = h$.*

Démonstration : Si $|h| = 1$, c'est évident. Supposons $|h| \geqq 2$. Soit $k = \max \{i / z'(i) \leqq h$ et $C_i(0.h) \neq C_i(\bar{0}.h)\}$; k existe car $|h| \geqq 2$. $E(k) \neq \varnothing$ car $h \in E(k)$; donc $k < l'$. Si $h \neq \min E(k)$ alors $z'(k+1) < h$, et $C_{k+1}(0.h) \neq C_{k+1}\bar{0}.h)$ [en effet, l'un des deux états $0.h$ et $\bar{0}.h$ est final et l'autre pas, par conséquent, l'égalité $C_{k+1}(0.h) = C_{k+1}(\bar{0}.h)$ impliquerait l'existence d'un entier $j \leqq k+1$ tel que l'un des deux états $0.z'(j)$ et $\bar{0}.z'(j)$ soit final et l'autre pas, $z'(j)$ distinguerait alors u et v, ce qui contredirait la minimalité de h]. Ceci est impossible, k étant maximal, donc $h = z'(k+1)$ et $k+1 \leqq l'$. ■

On en déduit l'algorithme 12. Il effectue le calcul de $d(u, v)$ en utilisant la suite $t'(i) = (0.z'(i), \bar{0}.z'(i))$ $(1 \leqq i \leqq l')$. F est une file d'éléments de la forme $(p, \bar{q}, k)$, où $(p, \bar{q}) = t'(i)$, $k = |z'(i)|$ $(1 \leqq i \leqq l')$.

ALGORITHME 12 : Fonction $d(u, v)$

```
Début
    INITIALISER (F); AJOUTER ((0, 0̄, 0), F);
    tant que F ≠ file vide faire
        début
            (p, q̄, k) ← TÊTE(F); SUPPRIMER (F);
            pour a variant de la première à la dernière lettre de A faire
                si COMPOSANTE (p.a) ≠ COMPOSANTE (q̄.a) alors
                    début
                        si (p.a, q̄.a) ∈ D alors retour (k);
                        AJOUTER ((p.a, q̄.a, k+1), F).
                        UNION (p.a, q̄.a)
                    fin {si}
        fin; {tant que}
```

retour (∞)
fin {fonction}

REMARQUE : Si l'on souhaite obtenir effectivement h, le mot le plus petit qui distingue u et v, il suffit de définir un tableau PERE et d'effectuer PERE $[p.a, \bar{q}.a] \leftarrow (p, \bar{q}, a)$ à chaque fois que l'élément $(p.a, \bar{q}.a, k+1)$ est placé dans la file. On peut alors facilement construire h à partir de PERE.

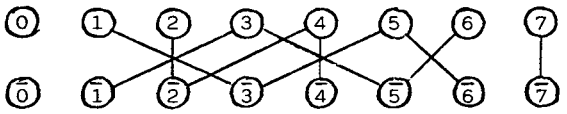
Exemple :

$A=\{a, b, c\}$, $u=cabacb$, $v=bacabc$

$\mathcal{S}(u)$: ⓪ c ① a ② b ③ a ④ c ⑤ b ⑥ ⑦

⓪ b ① a ② c ③ a ④ b ⑤ c ⑥ ⑦

Graphe



File

$(0, \bar{0}, 0)$

$(2, \bar{2}, 1)$ $(3, \bar{1}, 1)$ $(1, \bar{3}, 1)$

$(3, \bar{1}, 1)$ $(1, \bar{3}, 1)$ $(4, \bar{4}, 2)$ $(3, \bar{5}, 2)$ $(5, \bar{3}, 2)$

$(1, \bar{3}, 1)$ $(4, \bar{4}, 2)$ $(3, \bar{5}, 2)$ $(5, \bar{3}, 2)$ $(4, \bar{2}, 2)$ $(6, \bar{5}, 2)$

$(4, \bar{4}, 2)$ $(3, \bar{5}, 2)$ $(5, \bar{3}, 2)$ $(4, \bar{2}, 2)$ $(6, \bar{5}, 2)$ $(5, \bar{6}, 2)$

$(3, \bar{5}, 2)$ $(5, \bar{3}, 2)$ $(4, \bar{2}, 2)$ $(6, \bar{5}, 2)$ $(5, \bar{6}, 2)$ $(7, \bar{7}, 3)$

$(5, \bar{3}, 2)$ $(4, \bar{2}, 2)$ $(6, \bar{5}, 2)$ $(5, \bar{6}, 2)$ $(7, \bar{7}, 3)$

$(p, \bar{q}, k)=(3, \bar{5}, 2)$, $3.a=4$, $\bar{5}.a=\bar{7}$

$(4, \bar{7}) \in D \Rightarrow \underline{d(u, v)=2}$

PÈRE

		+			
		$2, \bar{2}$	$3, \bar{1}$	$1, \bar{3}$	$4, \bar{4}$
		$0, \bar{0}, a$	$0, \bar{0}, b$	$0, \bar{0}, c$	$2, \bar{2}, a$
+					
$3, \bar{5}$	$5, \bar{3}$	$4, \bar{2}$	$6, \bar{5}$	$5, \bar{6}$	$7, \bar{7}$
$2, \bar{2}, b$	$2, \bar{2}, c$	$3, \bar{1}, a$	$3, \bar{1}, b$	$1, \bar{3}, c$	$4, \bar{4}, a$

$\underline{h=aba}$

On remarque que, par exemple, $(2, \bar{4}, 2)$ n'a pas été placé dans la file car 2 et $\bar{4}$ étaient déjà dans la même composante connexe.

Mise en œuvre : On connaît une méthode optimale pour mettre en œuvre les opérations UNION et COMPOSANTE (*cf.* [AHU 74]). Elle permet d'effectuer une suite de $O(|Z|)$ opérations UNION et COMPOSANTE en temps $O(|S|\alpha(|S|))$, où α est une fonction dont la croissance est extrêmement faible : pour tout entier n, $\alpha(n)$ est le plus petit entier k tel que $F(k) \geqq n$, avec

$$F(0)=1 \qquad \text{et} \qquad F(i)=2^{F(i-1)}$$

pour $i>0$ ($\alpha(n) \leqq 5$ pour tout $n \leqq 2^{65536}$). En outre, la complexité spatiale de la méthode est linéaire.

PROPOSITION 13. — *Soient u et v deux mots de A*. Le calcul de d(u, v) par l'algorithme* 12 *est effectué en temps*

$$O(|A|(|u|+|v|)\alpha(|u|+|v|)),$$

sa complexité en espace est $O(|A|(|u|+|v|))$.

Démonstration : Soit N_1 et N_2 respectivement le nombre d'opérations UNION et COMPOSANTE effectuées par l'algorithme. L'opération UNION provoque l'ajout d'une arête entre deux sommets dont les composantes connexes sont distinctes, elle est donc réalisée au plus $|S.|-1$ fois. Par conséquent, le nombre d'éléments placés dans la file est inférieur ou égal à $|S|-1$. On a

$$N_1 \leqq |S|-1 \qquad \text{et} \qquad N_2 \leqq 2|A|(|S|-1).\ |S|=|u|+|v|+4,$$

d'où le résultat. ■

BIBLIOGRAPHIE

AHU 74. A. V. AHO, J. E. HOPCROFT et J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

CC 82. A. CARDON et M. CROCHEMORE, *Partitioning a graph in* $O(|A|\log_2|V|)$, Theor. Comput. Sci. vol. 19, 1982, p. 85-98.

He 84. J. J. HEBRARD, *Distances sur les mots. Application à la recherche de motifs*, Thèse de 3e cycle, Université de Haute-Normandie, 1984.

Hi 77. D. S. HIRSCHBERG, *Algorithms for the Longest Common Subsequence Problem*, J. Assoc. Comput. Mach., vol. 24, 1977, p. 664-675.

HS 77. J. W. HUNT et T. G. SZYMANSKI, *A Fast Algorithm for Computing Longest Common Subsequences*, Comm. ACM., vol. 20, 1977, p. 350-353.

Lo 82. LOTHAIRE, *Combinatorics on Words*, Addition-Wesley, Reading, Mass., 1982.

Mo 70. H. L. MORGAN, *Spelling Correction in Systems Programs*, Comm. ACM., vol. 13, 1970, p. 90-94.

MP 80. W. J. MASEK et M. S. PATERSON, *A Faster Algorithm Computing String Edit Distances*, J. Comput. and Sys. Sci., vol. 20, 1980, p. 18-31.

NKY 82. N. NAKATSU, Y. KAMBAYASHI et S. YAJIMA, *A Longest Common Subsequence Algorithm Suitable for Similar Test Strings*, Acta Informatica, vol. 18, 1982, p. 171-179.

Se 74. P. H. SELLERS, *An Algorithm for the Distance between two Finite Sequences*, J. Combinatorial Theory, Series A, vol. 16, 1974, p. 253-258.

Si 84. I. SIMON, *An Algorithm to Distingsh Words Efficiently by their Subwords* Communication at "Combinatorial Algorithms on words" conference Maratea (1984).

SK 83. D. SANKOFF et J. B. KRUSKAL, *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, Mass., 1983.

WF 74. R. A. WAGNER et M. J. FISCHER, *The String to String Correction Problem*, J. Assoc. Comput. Mach., vol. 21, 1974, p. 168-173.