RAIRO. INFORMATIQUE THÉORIQUE

MARTINE PAGET

Propriétés de complexité pour une famille d'algorithmes de Markov

RAIRO. Informatique théorique, tome 12, n° 1 (1978), p. 15-32

<http://www.numdam.org/item?id=ITA_1978__12_1_15_0>

R.A.I.R.O. Informatique théorique/Theoretical Computer Science
(vol. 12, n° 1, 1978, p. 15 à 32)

PROPRIÉTÉS DE COMPLEXITÉ POUR UNE FAMILLE D'ALGORITHMES DE MARKOV (*)

par Martine PAGET (1)

Communiqué par G. Ausiello

Résumé. — *Nous étudions les relations entre les algorithmes de Markov normaux, à étiquettes, à plusieurs mots et d'autres modèles abstraits (machine de Turing, RAM). Nous présentons une nouvelle machine abstraite appelée machine à mémoire associative permettant l'exécution d'un algorithme de Markov en temps et mémoire linéaire.*

INTRODUCTION

Les algorithmes de Markov (A. M.) ont été introduits par Markov [17] afin de résoudre des problèmes de décidabilité sur des mots. Par la suite, on les a mis en œuvre pour l'étude des langages de programmation tant pour une définition formelle de langages existants (Algol 60 [22]), que pour la conception et la définition de nouveaux langages (Snobol [9], Panon 1 B [6]).

Plus récemment, Aguzzi, Cesarini, Pinzani ont utilisé les algorithmes de Markov pour résoudre certains problèmes d'implémentation et de manipulation des arbres [1]. Mais l'utilisation des algorithmes de Markov et par suite de langages de programmation inspirés de ceux-ci s'est heurtée aux difficultés de leur implémentation et de leur exécution sur les ordinateurs actuels, en raison notamment de la lenteur de la recherche des occurrences de mots [7]. Diverses tentatives ont été faites pour améliorer la vitesse d'exécution d'un algorithme de Markov sur une machine classique. Ainsi, Katzenelson [14] impose certaines conditions aux règles de production pour obtenir un temps borné linéairement par la longueur du mot donné. Lagana, Leoni, Pinzani, Sprugnoli [12] obtiennent un résultat comparable en utilisant des algorithmes de Markov à pointeurs. C'est à ce problème de l'amélioration de la vitesse d'exécution d'un algorithme de Markov sur un ordinateur que nous nous intéressons ici. Pour cela, nous avons été amenés à introduire un nouveau modèle abstrait appelé MMA (machine à mémoire associative).

(*) Reçu mai 1977, révisé octobre 1977.

(1) Institut de Programmation, Université Pierre-et-Marie-Curie, Paris Cedex.

Le point de départ de notre travail est l'étude des relations entre différents modèles abstraits de calcul, du point de vue du temps d'exécution et de l'occupation mémoire, comme celles qui ont été établies entre les machines de Turing et les Random Access Machines ([3, 8] par exemple). Les deux premiers paragraphes sont consacrés à l'extension de cette étude à différents types d'algorithmes de Markov.

Après avoir donné la définition d'un algorithme de Markov et des mesures de complexité qui lui sont associées (mémoire et temps d'exécution à ne pas confondre avec temps d'exécution sur un autre modèle qui est en fait le temps de simulation), nous étudions au paragraphe 1 les relations entre machines de Turing, Random Access Machine et algorithme de Markov. Nous vérifions que ces relations sont polynomiales ce qui permet le transfert de certains résultats d'un modèle à un autre. Nous utilisons dans cette partie un algorithme de Morris et Pratt [19] [recherche d'une occurrence d'un mot y dans un mot x en temps $O(|x|+|y|)$] qui fait diminuer fortement le facteur multiplicatif constant du résultat dans la simulation d'un algorithme de Markov sur une Random Access Machine. Malgré tout, ces résultats n'améliorent pas de façon sensible (sauf dans certains cas), la vitesse d'exécution d'un algorithme de Markov sur une machine classique. En effet, nous obtenons le résultat suivant dont l'importance vient de ce qu'une Random Access Machine est le modèle le plus proche des ordinateurs actuels :

« si un algorithme de Markov fonctionne en temps T, alors il existe une Random Access Machine sur laquelle on simule l'algorithme de Markov en temps $O(T^2 \operatorname{Log} T)$ ».

Nous étudions au paragraphe 2 les liens entre les différents types d'algorithmes de Markov.

L'introduction d'un nouveau type :

« l'A.M. à k mots », outre qu'il permet de simuler facilement une machine de Turing à k bandes, fait passer très simplement d'un A.M. à étiquettes (d'où les instructions de Snobol sont inspirées) à un A.M. normal, alors que le passage direct est ardu [10].

Dans le but d'améliorer la vitesse d'exécution d'un algorithme de Markov sur une machine, nous introduisons au paragraphe 3 un modèle abstrait appelé MMA (machine à mémoire associative). Dans les travaux de Katzenelson et de Lagana, Leoni, Pinzani, Sprugnoli, la recherche de l'occurrence reste classique. Nous avons essayé de ne pas modifier la structure d'exécution d'un algorithme de Markov et de trouver une solution valable pour tous les types d'algorithmes de Markov. Il fallait donc trouver une recherche d'occurrence plus rapide. L'utilisation des mémoires associatives a permis cette amélioration. L'intérêt de ces mémoires est d'avoir accès aux mots mémoires en se servant non pas de leur adresse mais de leur contenu. Ainsi on peut repérer (ou modifier) tous les mots ayant un même contenu à l'aide d'une seule opéra-

tion effectuée en parallèle sur toute la mémoire. Nous établissons les relations entre machine à mémoire associative et les autres modèles. Les possibilités des MMA permettent la recherche d'une occurrence d'un mot y dans un mot x en temps $O(|y|)$ ce qui fournit le résultat principal de notre travail (proposition 11) :

« si un algorithme de Markov (normal ou à étiquettes) fonctionne en temps T et en mémoire L, alors il existe une MMA sur laquelle on simule l'algorithme de Markov en temps $O(T)$ et en mémoire $O(L)$ ».

On peut en conclure que l'utilisation des mémoires associatives, dont le coût encore trop élevé a limité l'utilisation à des problèmes ponctuels (pagination, contrôle aérien...), pourrait faciliter l'implémentation des algorithmes de Markov et des langages adaptés aux traitements de textes.

1. RELATIONS ENTRE ALGORITHMES DE MARKOV, MACHINES DE TURING ET RANDOM ACCESS MACHINES

Nous donnons dans cette section la définition d'un algorithme de Markov et nous étudions les relations entre algorithme de Markov, machines de Turing et RAM.

DÉFINITIONS : *Un algorithme de Markov normal* (*A.M.*) d'alphabet A et de longueur p est une structure algébrique partielle :

$M_n = \langle \{1, \ldots, p\}, A^*; \pi; F \rangle$
où π est une application de $1, \ldots, p$ dans $A^* \times A^*$
et F un sous-ensemble de $\{1, \ldots, p\}$.

NOTATIONS : si $\pi(k) = \langle u, v \rangle$ alors suivant que

$$k \notin F \text{ ou } k \in F \text{ on note } : k : u \to v \text{ ou } k : u \to .v$$

on dit que k est *l'étiquette de la production*, $u \to v$ une *production simple*, $u \to .v$ une *production terminale*.

Un *calcul* dans l'algorithme se déroule de la manière suivante :

pour la donnée w, on examine successivement les productions à partir de la 1re production :

s'il n'existe pas de production applicable à w (c'est-à-dire telle que u ait une occurrence dans w) le calcul s'arrête, sinon on applique la première production trouvée en remplaçant l'occurrence u la plus à gauche dans w par v, ce qui donne un mot w'. Si la production appliquée est terminale, le calcul s'arrête; sinon on recommence à chercher une production applicable à w' à partir de la 1re production.

Il s'agit là d'une machine déterministe dont le calcul peut éventuellement se poursuivre indéfiniment.

Exemple : L'algorithme suivant $M_1 = \langle \{ 1, \ldots, 10 \}, \{ a, b, \alpha, \beta \}^*; \pi; 4 \rangle$ calcule l'image miroir d'un mot de $\{ a, b \}^*$:

1 : $\alpha\alpha \rightarrow \beta$
2 : $\beta\alpha \rightarrow \beta$
3 : $\beta a \rightarrow a \beta$
4 : $\beta b \rightarrow b \beta$
5 : $\beta \rightarrow .$
6 : $\alpha ab \rightarrow b \alpha a$
7 : $\alpha ba \rightarrow a \alpha b$
8 : $\alpha bb \rightarrow b \alpha b$
9 : $\alpha aa \rightarrow a \alpha a$
10 : $\rightarrow \alpha$

Le temps d'exécution d'une production est pris égal à l'unité.

Le temps d'exécution d'un calcul est le nombre de productions exécutées au cours du calcul.

L'occupation mémoire d'un calcul est la longueur du mot le plus long apparaissant au cours du calcul.

Soient T et L deux fonctions récursives de **N** dans **N**, on dit qu'un A.M. (une machine de Turing, une RAM) *fonctionne* ou *opère* en temps (resp. en mémoire) T (resp. L) si pour une donnée de longueur n, le temps de calcul (resp. l'occupation mémoire) est borné par $T(n)$ [resp. $L(n)$].

Dans tout ce qui suit, nous avons adopté le coût logarithmique pour les mesures de complexité des RAM; les machines de Turing sont à une bande sauf avis contaires et A.M. signifie algorithme de Markov normal.

PROPOSITION 1 : *Si une machine de Turing $\mathfrak{T}$ opère en temps T et en mémoire L alors on peut construire un A.M. sur lequel on simule le fonctionnement de $\mathfrak{T}$ et qui opère en temps T et en mémoire L.*

Preuve : L'A.M. n'est rien d'autre que la grammaire associée à l'automate de Turing.

Q.E.D.

PROPOSITION 2 : *Si un A.M. $\mathfrak{M}$ fonctionne en temps T et en mémoire L et si $n \leqq T(n)$ alors on peut construire une machine de Turing sur laquelle on simule le fonctionnement de $\mathfrak{M}$ et qui opère en temps T^2 et en mémoire L.*

Preuve : Pour une donnée de longueur n, la simulation de chaque production $u_i \rightarrow v_i$ exige la recherche d'occurrence d'un mot de longueur $| u_i |$ dans un mot de longueur $\leqq L(n)$ ce qui nécessite au plus $k \times L(n)$ comparaisons, k dépendant de l'A.M. considéré. Comme il y a $T(n)$ productions appliquées, le temps obtenu est au plus $k \times L(n) \times T(n)$ ce qui est borné par $k\, T(n)^2$ si $n \leqq T(n)$.

La machine de Turing a donc un nombre d'opérations borné par T^2 d'après le théorème d'accélération linéaire pour les machines de Turing [12]. L'occurpation mémoire est la même que celle de l'A.M.

Q.E.D.

La borne T^2 est la meilleure que l'on puisse trouver, en effet : (résultat communiqué par A. Slisenko, Léningrad) :

PROPOSITION 3: *Il existe un A. M. fonctionnant en temps $T(n) = n$ qui ne peut être simulé en moins de $T'(n) = n^2$ sur une machine de Turing.*

Preuve : Considérons le langage $L = \{ wc\tilde{w} \mid w \in \{ 0, 1 \}^* \}$. Ce langage est reconnu en temps linéaire par l'A.M. suivant :

$$0c0 \to c$$

$$1c1 \to c$$

$$0c1 \to .m$$

$$1c0 \to .m$$

$$c \to .$$

(le mot donné $\in L$ si le résultat est le mot vide).

Or on sait [13] que ce langage L ne peut être accepté en moins de $T(n) = n^2$ opérations par une machine de Turing à une bande.

Q.E.D.

PROPOSITION 4 : *Si un A.M. $\mathfrak{M}$ fonctionne en temps T et en mémoire L et si $n \leqq T(n)$, alors on peut construire une RAM qui simule le fonctionnement de $\mathfrak{M}$ et qui opère en temps $O(T^2 \log T)$ et en mémoire $O(L)$.*

Preuve : Dans la preuve de cette proposition, nous avons essayé de minimiser le facteur constant multiplicatif du résultat, ce qui d'un point de vue pratique réduit fortement le temps d'exécution dans de nombreux cas. Soit l'A.M. :

$$\begin{aligned} u_1 &\to v_1 \\ &\vdots \\ u_p &\to v_p \end{aligned}$$

Le nombre d'opérations sur une RAM nécessaires à la recherche d'occurrence avec un algorithme ordinaire est de l'ordre de $(|u|_1 + \ldots + |u|_p) \times L \operatorname{Log} L$. Avec l'algorithme de Morris et Pratt [11] on obtient

$$(|u|_1 + \ldots + |u|_p + p\,L) \times \operatorname{Log} L.$$

Si $u_k = b_1 \ldots b_s$, on définit la fonction $f_{u_k}(i)$ = le plus grand $r < i$ tel que $b_1 \ldots b_r$ est un suffixe de $b_1 \ldots b_i$ (i.e. : $b_1 \ldots b_r = b_{i-r+1} \ldots b_i$), 0 si r n'existe pas.

On commence par calculer cette fonction pour chaque u_k ($k = 1, p$). Pour ce calcul on exécute $k \mid u_i \mid$ opérations (pour la justification de ce temps *voir* [3]) dont le temps est borné pour chacune par

$$\text{Log}\,(\mid u_1 \mid + \mid u_2 \mid + \ldots + \mid u_p \mid),$$

(lorsqu'il y a indirection $\mid u_1 \mid + \ldots + \mid u_p \mid$ est la plus grande adresse dont on peut se servir, *voir* ci-dessous l'utilisation des registres de la RAM). Le temps total de ce calcul est borné par $k \sum_{i=1}^{p} \mid u_i \mid \text{Log} \sum_{i=1}^{p} \mid u_i \mid$, c'est une constante liée à l'A.M. considéré.

Utilisation des registres de la RAM dans la simulation d'un A.M. :

Adresse	Utilisation ou contenu
1	valeur de f_{u_1}
$\vdots$	
$\mid u_1 \mid$	
$\mid u_1 \mid + 1$	
$\vdots$	
$\mid u_1 \mid + \mid u_2 \mid$	
$\vdots$	
$\mid u_1 \mid + \ldots + \mid u_p \mid$	
$\mid u_1 \mid + \ldots + \mid u_p \mid + 1$	registres auxiliaires
$\vdots$	
$\mid u_1 \mid + \ldots + \mid u_p \mid + k$	
$\mid u_1 \mid + \ldots + \mid u_p \mid + k + 1$	registres contenant chacun une lettre du mot u_i lorsque l'on calcule f_i ou lors du test de la i-ième production ou bien contenant le mot v_j à substituer
$\vdots$	
$x = \mid u_1 \mid + \ldots + \mid u_p \mid + k + \max\left(\max_{i=1}^{p} \mid u_i \mid, \max_{i=1}^{p} \mid v_i \mid \right)$	
$x + 1$	mot donné
$\vdots$	
$x + L(n)$	

On simule ensuite l'A.M. : La phase de recherche de l'occurrence se fait sans retour arrière dans le mot donné à l'aide de la fonction f [12]. Cette recherche se fait en $L(n)+k$ opérations, chacune ayant un temps d'exécution borné par $\text{Log}(L(n))+k_1$; il en est de même pour la phase de remplacement de u_i par v_i qui comporte au plus $L(n)+k_2$ opérations de décalage ou de substitution chacune ayant un temps d'exécution borné par $\text{Log}(L(n))+k_3$. Au total la simulation de l'application d'une production se fait en un temps borné par $k(L(n)\,\text{Log}\,L(n)) \leqq k\,T(n)\,\text{Log}\,T(n)$ si $n \leqq T(n)$. S'il y a $T(n)$ productions appliquées, le temps de simulation de l'A.M. par une RAM est borné par $O(T^2\,\text{Log}\,T)$ si $T(n) \geqq n$.

Il y a $L(n)+k$ registres utilisés chacun contenant un nombre dont la longueur est bornée par $\text{Log}(a)$ si $a = \text{Card}$ (alphabet) sauf un registre servant pour l'indirection contenant un nombre dont la longueur est bornée par $\text{Log}(n)$. L'occupation mémoire est donc

$$L(n)+\text{Log}\,L(n)+k \leqq p\,L(n),\ p \geqq 2.$$

Q.E.D.

2. RELATIONS ENTRE DIFFÉRENTS TYPES D'ALGORITHMES DE MARKOV

Nous nous intéressons dans ce deuxième paragraphe à différents types d'algorithmes de Markov (A.M. à k mots, A.M. à étiquettes). Les A.M. à étiquettes ont été introduits par Galler [10] et ont servi de modèle pour la définition du langage de programmation Snobol [9]. Nous introduisons les A.M. à k mots pour résoudre certains problèmes plus facilement (par exemple pour les problèmes nécessitant un comptage, un 2^e mot servant de compteur), comme avaient été introduits les machines de Turing à k bandes. Nous établissons les relations entre ces différents algorithmes et les autres modèles. Notons que la simulation d'un A.M. à étiquette sur un A.M. normal est facilitée par l'utilisation d'une simulation intermédiaire sur un A.M. à 2 mots.

Algorithmes de Markov à étiquettes [10]

C'est cette forme d'algorithme de Markov qui a servi de modèle au langage de programmation Snobol [9].

DÉFINITIONS : *Un algorithme de Markov à étiquettes* d'alphabet A et de longueur p est une structure algébrique partielle :

$$M_e = \langle \{1, \ldots, p\}, A^*; \pi; F \rangle,$$

où π est une application de

$$\{1, \ldots, p\} \quad \text{dans } (A^*)^2 \times \{1, \ldots, p\}^2 \qquad \text{et} \qquad F \subseteq \{1, \ldots, p\}.$$

Les productions sont de la forme :

$$k : \quad u \to (.)v,\ i_k, j_k$$

Lorsqu'une production k est appliquée (une production est applicable dans les mêmes conditions que pour un A.M. normal), la recherche d'une nouvelle production applicable se fait à la production d'étiquette i_k; si la production n'est pas applicable on cherche une nouvelle production applicable à la production d'étiquette j_k.

On utilise les mêmes définitions du temps d'exécution et de l'occupation mémoire que pour les A.M. normaux.

Avant de passer à la simulation d'un algorithme de Markov à étiquettes sur un algorithme de Markov normal nous allons définir les algorithmes de Markov à k mots qui facilitent la simulation d'un algorithme de Markov à étiquettes.

Algorithmes de Markov à k mots

DÉFINITION : *Un algorithme de Markov à k mots* d'alphabet A et de longueur p est une structure algébrique partielle $M_k = \langle \{ 1, \ldots, p \}, A^*; \pi; F \rangle$, où π est une application de $\{ 1, \ldots, p \}$ dans $(A^*)^{2k}$.

Les productions sont de la forme :

$$e : \begin{pmatrix} u_{e_1} \\ \vdots \\ u_{e_k} \end{pmatrix} \rightarrow (\cdot) \begin{pmatrix} v_{e_1} \\ \\ v_{e_k} \end{pmatrix}$$ et sont appelées k-productions.

Le fonctionnement d'un A.M. à k mots est le même que celui d'un A.M. normal en posant qu'une k-production $(u_1, \ldots, u_k) \rightarrow (v_1, \ldots, v_k)$ est applicable au k-mot donné $(w_1, \ldots, w_k)$ lorsque pour tout i, $1 \leqq i \leqq k$, u_i a une occurrence dans w_i (alors on choisit l'occurrence la plus à gauche pour la remplacer par v_i).

Exemple : L'A.M. à 2 mots suivant calcule l'image miroir d'un mot de A^*. La donnée est sur le 1er mot, le résultat sur le 2e.

Pour tout $a \in A$:

$$0 : \begin{pmatrix} \alpha a \end{pmatrix} \rightarrow \begin{pmatrix} a \alpha \\ a \end{pmatrix}$$

$$1 : \begin{pmatrix} \alpha \end{pmatrix} \rightarrow (\cdot) \begin{pmatrix} \end{pmatrix}$$

$$2 : \begin{pmatrix} \end{pmatrix} \rightarrow \begin{pmatrix} \alpha \end{pmatrix}$$

Le temps d'exécution d'une k-production est pris égal à l'unité.

Le temps d'exécution d'un calcul est le nombre de k productions exécutées au cours du calcul.

L'occupation mémoire d'un calcul est la longueur du k-mot le plus long apparaissant au cours du calcul.

(La longueur d'un k mot est la somme des longueurs des k-mots.)

PROPOSITION 5 : *Si un A.M. à k mots $\mathfrak{M}_k$ fonctionne en temps T et en mémoire L et si $T(n) \geqq n$, il existe un A.M. normal $\mathfrak{M}_1$ qui lui est équivalent et qui fonctionne en temps $O(T^2)$ et en mémoire $O(L)$.*

Preuve : Soient p le nombre de productions de $\mathfrak{M}_k$, $\bigstar$ un symbole n'appartenant pas à l'alphabet de $\mathfrak{M}_k$. Si la donnée est $(a_1, \ldots, a_k)$, on parcourt le mot $a_1 \bigstar a_2 \bigstar \ldots \bigstar a_k$ dans $\mathfrak{M}_1$ à la recherche d'une production $(u_{i_1}, \ldots, u_{i_k}) \rightarrow (v_{i_1}, \ldots, v_{i_k})$ applicable à $(a_1, \ldots, a_k)$ c'est-à-dire à la recherche d'une occurrence de u_{i_1} dans a_1, de u_{i_2} dans a_2, ..., de u_{i_k} dans a_k pour $i = 1, p$; on marque successivement u_{i_1} par $\bar{u}_{i_1}$ (les symboles barrés n'appartenant pas à l'alphabet de $\mathfrak{M}_k$), u_{i_2} par $\bar{u}_{i_2}$ etc. Si pour i donné, la production i n'est pas applicable, on remet la donnée dans son état initial (i.e. on remplace $\bar{u}_{i_1}$ par u_{i_1} etc.) et l'on passe au test de la production $i+1$, sinon on effectue le remplacement de $\bar{u}_{i_1}$ par v_{i_1}, ..., $\bar{u}_{i_k}$ par v_{i_k} (*voir* [20] pour la simulation complète).

Le temps de simulation d'une production de $\mathfrak{M}_k$ est borné par $p \times 2 \times L(n) + k$ (parcours aller et retour du mot donné).

S'il y a $T(n)$ production appliquées dans $\mathfrak{M}_k$, le temps de simulation dans $\mathfrak{M}_1$ est borné par $K \times L(n) \times T(n) \leqq K \times T^2(n)$ si $n \leqq T(n)$.

L'occupation mémoire est la même.

Q.E.D.

PROPOSITION 6 : *Si une machine de Turing à k bandes $\mathfrak{T}$ fonctionne en mémoire L et en temps T alors il existe un A.M. à k mots sur lequel on simule le fonctionnement de $\mathfrak{T}$ en mémoire $O(L)$ et en temps $O(T)$.*

Preuve : Il suffit d'associer à chaque bande un k-mot et la simulation se déduit du cas à une bande.

Q.E.D.

PROPOSITION 7 : *Si un A.M. à k mots $\mathfrak{M}_k$ fonctionne en temps T et en mémoire L il existe une machine de Turing à k bandes sur laquelle on simule le fonctionnement de $\mathfrak{M}_k$ en mémoire L et en temps T^2.*

Preuve : On procède de la même manière que pour simuler un A.M. normal, chaque bande contenant un k-mot, la recherche des occurrences se faisant en parallèle sur les k bandes. Il suffit de synchroniser les phases de recherche et de remplacement d'occurrences sur les différentes bandes ce qui ne modifie pas l'évaluation en temps et en mémoire faite dans le cas des machines de Turing à une bande (*cf.* prop. 2).

Q.E.D.

PROPOSITION 8 : *Si un A.M. à étiquettes $\mathfrak{M}_e$ fonctionne en temps T et en mémoire L, il existe un A.M. $\mathfrak{M}_2$ à 2 mots qui simule le fonctionnement de $\mathfrak{M}_e$ en temps $O(T)$ et en mémoire $O(L)$.*

Preuve : Toute production $k : u \to v$, i, j de $\mathfrak{M}_e$ *est remplacée par les deux 2-productions :*

$$2k : \begin{pmatrix} u \\ 2k \end{pmatrix} \to \begin{pmatrix} v \\ 2i \end{pmatrix}$$

$$2k+1 : \begin{pmatrix} \\ 2k \end{pmatrix} \to \begin{pmatrix} \\ 2j \end{pmatrix}$$

Si w est la donnée dans $\mathfrak{M}_e$, $\begin{pmatrix} w \\ 1 \end{pmatrix}$ est la donnée dans $\mathfrak{M}_2$. Le deuxième mot simule le contrôle de l'étiquette de branchement. Si p est le nombre de productions de $\mathfrak{M}_e$, l'application d'une production dans $\mathfrak{M}_e$ nécessite l'application d'au plus p productions dans $\mathfrak{M}_2$. Si $\mathfrak{M}_e$ fonctionne en temps T, $\mathfrak{M}_2$ fonctionne en temps $O(T)$. L'occupation mémoire est $O(L)$ dans $\mathfrak{M}_2$.

Q.E.D.

COROLLAIRE : *Si un A.M. à étiquettes $\mathfrak{M}_e$ fonctionne en temps T et en mémoire L et si $n \leqq T(n)$, il existe un A.M. normal qui simule le fonctionnement de $\mathfrak{M}_e$ en temps $O(T^2)$ et en mémoire $O(L)$.*

Immédiat d'après les propositions 5 et 8.

3. MACHINES A MÉMOIRES ASSOCIATIVES (MMA)

Nous allons définir un nouveau modèle abstrait en nous inspirant des caractéristiques des processeurs associatifs [4, 18]. Puis nous établirons les relations entre les MMA et les autres modèles; ces résultats nous permettent de conclure à l'efficacité de l'utilisation des mémoires associatives pour exécuter un algorithme de Markov.

DÉFINITION : *Une MMA* (machine à mémoire associative) est un programme fini opérant sur une suite infinie de registres contenant un symbole appartenant à A (alphabet de la MMA). Les registres sont indicés en ordre croissant à partir de 0. A chaque registre est associé son état. Un registre peut prendre deux états différents : état actif noté 1, ou état inactif noté 0. Au début d'un programme, tous les registres sont supposés être dans l'état inactif. On notera $\langle R(i, t) \rangle = (\alpha, \sigma)$ pour signifier que le contenu du registre d'indice i est α à l'instant t et qu'il se trouve dans l'état σ. A un instant t, un nombre fini m de registres est utilisé.

Les opérations n'opèrent que sur les registres d'indice $\leqq m$ ou éventuellement sur le registre $m+1$ (alors $m := m+1$). Chaque opération fait passer de l'instant t à $t+1$. Si $\langle R(i, t) \rangle = (\alpha, n)$ alors $p_1^2(\langle R(i, t) \rangle) = \alpha$ et $p_2^2(\langle R(i, t) \rangle) = n$.

Si la donnée (un mot dans A^*) est de longueur m, elle se trouve dans les m premiers registres. On trouvera en annexe le *tableau des instructions.*

Le temps d'exécution de chaque instruction est pris égal à l'unité. En effet dans le cas concret des processeurs associatifs, les opérations se font en parallèle pour tous les mots de la mémoire et le temps est indépendant de leur place en mémoire.

Le temps d'exécution d'un programme est donc le nombre d'opérations effectuées au cours du calcul.

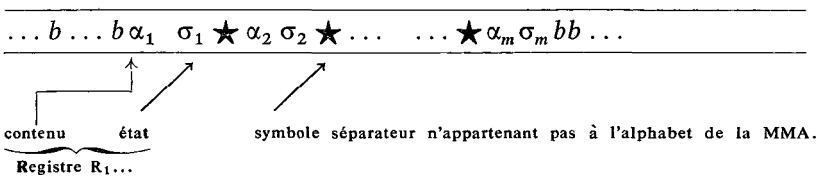
L'occupation mémoire est le nombre maximum de registres utilisés au cours du calcul.

Relations entre les MMA et les autres modèles

Les propositions et corollaires ci-dessous montrent en outre l'équivalence du point de vue même puissance de calcul entre les MMA et les autres modèles.

PROPOSITION 9 : *Si une MMA fonctionne en temps T et en mémoire L et si $n \leqq T(n)$ alors il existe une machine de Turing qui simule le fonctionnement de la MMA en temps T^2 et en mémoire L.*

Preuve : Le contenu des registres et leur état sont représentés de la manière suivante :



La simulation de chaque opération associative se fait en parcourant la bande donc en un temps borné par $O(L(n))$.

Comme il y a $T(n)$ opérations à simuler, on obtient un temps d'exécution borné par $k \times T(n) \times L(n) \leqq k\, T^2(n)$ si $n \leqq T(n)$, c'est-à-dire borné par $T^2(n)$ d'après le théorème d'accélération linéaire pour les machines de Turing. L'occupation mémoire est de l'ordre de $3\,L(n)+k$ qui peut se ramener à $L(n)$ par codage approprié.

Q.E.D.

D'après la proposition 1 et la proposition précédente on a :

COROLLAIRE : *Si une MMA fonctionne en temps T et en mémoire L, il existe un A.M. qui simule le fonctionnement de la MMA en temps $O(T^2)$ et en mémoire $O(L)$.*

PROPOSITION 10 : *Si une MMA fonctionne en temps T et en mémoire L et si $n \leqq T(n)$ alors il existe une RAM qui simule le fonctionnement de la MMA en temps $O(T^2 \operatorname{Log} T)$ et en mémoire $O(L)$.*

Preuve : Soient R_1, R_2 ... les registres de la MMA.
Utilisation des registres de la RAM dans la simulation d'une MMA :

Adresse	Contenu
1 ⋮ K	registres auxiliaires
$K+1$	contenu du registre R_1 de la MMA
$K+2$	état du registre R_1
⋮	⋮
$K+2i-1$	contenu de R_i
$K+2i$	état du registre R_i
⋮	⋮
$K+2m$	état du registre R_m
$K+2m+1$	Registre « fin » contenant un symbole ★ n'appartenant pas à l'alphabet de la MMA

Pour la simulation de chaque type d'opérations associatives, on accède successivement à tous les registres simulant la mémoire associative et ses états. On effectue donc au plus $k_1 \times L(n)$ opérations, chacune ayant un temps soit constant (accès direct) soit borné par $k_2 \times \text{Log } L(n)$ (accès indirect).

Le temps de simulation d'une instruction est borné par

$$k_3 \times L(n) \times \text{Log}(L(n));$$

s'il y a $T(n)$ opérations, le temps de simulation sur la RAM sera borné par $O(T \times L \times \text{Log } L) \leqq O(T^2 \text{ Log } T)$ si $n \leqq T(n)$.

L'occupation mémoire est bornée par $k_1 \times (L(n) + \text{Log } L(n)) \leqq k_2 L(n)$.

Q.E.D.

PROPOSITION 11 : *Si un A.M. fonctionne en temps T et en mémoire L alors il existe une MMA qui simule l'A.M. et qui opère en temps O (T) et en mémoire O (L).*

Preuve : Soit l'A.M. :

$$u_1 \to v_1$$
$$\vdots$$
$$u_p \to v_p$$

Les n premiers registres de la MMA contiennent au début de la simulation la donnée de longueur n; le programme de simulation est de la forme : en mettant successivement pour chaque $i = 1, p$:

• recherche de u_i ($u_i = u_{i_1} \ldots u_{i_l}$) :

RECHi ACT u_{i_1}
BNA RECH$i+1$ — si pas de mot actif (si u_{i_1} n'existe pas) recherche de l'occurrence de u_{i+1} à l'adresse RECH$i+1$
DAD
ACTA u_{i_2} — recherche de l'occurrence de u_i
BNA RECH$i+1$
⋮
ACTA u_{i_l}
BNA RECH$i+1$
SAG — sélecter l'occurrence la plus à gauche

• remplacement de u_i par $v_i = v_{i_1} \ldots v_{i_k}$:

On pose $x_i = |u_i| - |v_i|$.

Cas 1 : $x_i > 0$, il faut « tasser » le mot de x positions

PAD
DMAG
⋮
DMAG
(x fois)

Cas 2 : $x_i = 0$, le remplacement peut être immédiat.

Cas 3 : $x_i < 0$ il faut « pousser » de x positions :

PAD
DMAD
⋮
DMAD
(x fois)
SAG
DAG

Le remplacement dans tous les cas se fait par :

ECR v_{i_k}
DAG
ECR $v_{i_{k-1}}$
DAG
⋮
ECR v_{i_1}
B RECHI ou STOP — branchement à la simulation de la production ou arrêt suivant que la production i est terminale ou non.

Pour chaque production u_i appliquée, on effectue au plus

$$\underset{\substack{\text{recherche}\\ \text{de l'occurence}}}{3|u_i|} + \underbrace{3+x}_{\substack{\text{"pousser"}\\ \text{ou}\\ \text{"tasser"}}} + \underset{\text{écriture}}{2|v_i|} = O(|u_i|+|v_i|) \text{ instructions}$$

Il faut rajouter les tests de recherche de u_1 à u_{i-1} ce qui est fait en temps borné par :

$O(|u_i|+\ldots+|u_{i-1}|)$. Chaque production est donc simulée en temps borné par une constante dépendante de l'algorithme simulé mais non de la donnée. Le temps de simulation est donc borné par $k \times T(n)$.

L'occupation mémoire est du même ordre que celle de l'A.M.

Q.E.D.

COROLLAIRE : *Si une machine de Turing $\mathfrak{T}$ fonctionne en temps T et en mémoire L alors il existe une MMA qui simule le fonctionnement de $\mathfrak{T}$ en temps $O(T)$ et en mémoire $O(L)$.*

Ce corollaire est immédiat d'après la proposition 1 et la proposition précédente.

PROPOSITION 12 : *Si un A.M. à k mots $\mathfrak{M}_k$ fonctionne en temps T et en mémoire L alors il existe une MMA qui simule le fonctionnement de $\mathfrak{M}_k$ en temps $O(T)$ et en mémoire $O(L)$.*

Preuve : On commence par disjoindre les alphabets auxquels appartiennent chaque symbole des k mots; soient $A_1, \ldots, A_k$ ces alphabets;

$$A = A_1 \cup A_2 \ldots \cup A_k;$$

les $n_1+\ldots+n_k$ premiers registres de la MMA contiennent le k mot donné (de longueur $n_1+\ldots+n_k$). La simulation de $\mathfrak{M}_k$ s'effectue de la manière suivante :

– pour chaque production i :

$$\begin{pmatrix} u_{i_1} \\ \vdots \\ u_{i_k} \end{pmatrix} \rightarrow (.) \begin{pmatrix} v_{i_1} \\ \vdots \\ v_{i_k} \end{pmatrix}$$

pour $l = 1, k$:

recherche de u_{i_l} : on procède de la manière décrite pour la recherche d'occurrence dans la simulation d'un A.M. normal; les alphabets de chaque k-mot étant disjoints il n'y a pas de confusion possible, c'est-à-dire pas de découverte d'occurrence de u_{i_l} ailleurs que dans le l-ième mot du k mot donné.

• Si l'occurrence est trouvée :

– si $l < k$ on remplace le dernier symbole de u_{i_l} par m_{l_k} (une marque quelconque n'appartenant pas à l'alphabet A) et l'on passe à la recherche de u_{l+1},

– si $l = k$, on effectue le remplacement de chaque u_{i_l} par v_{i_l} (pour l décroissant de k à 1) de la même manière que pour un A.M. normal, l'occurrence à remplacer étant repérée par m_{i_k}; puis l'on retourne à la simulation de la 1re production ou l'on s'arrête suivant que la production est terminale ou non;

• si l'occurrence n'est pas trouvée : on remplace chaque m_{j_k} ($j = 1$ à $l-1$) par le u_{j_k} correspondant et l'on passe à la simulation de la production suivante.

Comme pour la simulation d'un A.M. normal, le temps pour chaque production ne dépend pas de la longueur de la donnée mais est une constante de l'algorithme. Donc si $\mathfrak{M}_k$ fonctionne en temps T, la MMA simulant $\mathfrak{M}_k$ fonctionne en temps $k \times T$. L'occupation mémoire est identique à quelques symboles auxiliaires près.

Q.E.D.

COROLLAIRE : *Si une machine de Turing à k bandes $\mathfrak{T}_k$ ou un A.M. à étiquettes $\mathfrak{M}_e$ fonctionne en temps T et en mémoire L alors il existe une MMA qui simule le fonctionnement de $\mathfrak{T}_k$ ou de $\mathfrak{M}_e$ en temps $O(T)$ et en mémoire $O(L)$.*

C'est immédiat d'après la proposition précédente, la proposition 6 et la proposition 8.

CONCLUSION ET PERSPECTIVES

Nous avons complété l'étude des relations entre différents modèles abstraits de calcul. Après avoir montré que les algorithmes de Markov peuvent être utilisés concrètement d'une manière raisonnable (au sens temps et traitements raisonnables), on peut envisager la poursuite de l'étude des algorithmes de Markov non pas d'une manière purement théorique mais avec l'espoir d'apporter quelques solutions pratiques, en particulier aux traitements des textes, à la reconnaissance des langages [16]. Il serait aussi intéressant d'étudier les classes de complexité de fonctions calculables et de langages obtenues en calculant ces fonctions (ou en reconnaissant ces langages) sur un algorithme de Markov et comparer les résultats obtenus aux autres types de hiérarchies (hiérarchie de Grzegorczyck [11], de Ritchie [21], de Axt [2]).

BIBLIOGRAPHIE

1. G. AGUZZI, F. CESARINI et R. PINZANI, *Automatic Tree Structures Processing* in Proceedings of Colloque de Lille : Les arbres en Algèbre et en programmation, Lille, 1976.
2. P. AXT, *On a Subrecursive hierarchy and Primitive Recursive Degrees*, Trans. Amer. Math. Soc., vol. 92, 1959, p. 85-105.
3. A. V. AHO, J. E. HOPCROFT et J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.
4. P. A. BEAVEN et D. W. LEWIN, *An Associative Parallel Processing System for Non Numerical Computations*, The computer Journal, vol. 15, n° 4, 1972, p. 343-349.

5. M. BLUM, *A Machine Independant Theory of the Complexity of Recursive Functions*, J. Assoc. Comp. Mach., vol. 14, 1967, p. 322-336.
6. A. CARACCIOLO DI FORINO, L. SPANEDDA et N. WOLKENSTEIN, *Panon 1.B: a Programming Language for Symbol Manipulation*, Calcolo, 1966, p. 245-255.
7. A. CARACCIOLO DI FORINO, *Generalized Markov Algorithms and Automata*, Automata theory, ed., CAIANIELLO, Academic Press New York.
8. S. A. COOK et R. A. RECKOW, *Time Bounded Random Access Machines*, J. Comp. System. Sc. vol. 7, 1973, p. 354-375.
9. D. J. FARBER, R. E. GRISWOLD et I. P. POLONSKY, *Snobol, a String Manipulation Language*, J. Assoc. Comp. Mach., vol. 11, 1964, p. 21-30.
10. B. A. GALLER et A. J. PERLIS, *A view of Programming Languages*, Addison Wesley, 1970.
11. A. GRZEGORCZYCK, *Some Classes of Recursive Functions*, Rozprawy Matematyczne, vol. 4, Warsaw, 1953, p. 1-45.
12. J. HARTMANIS, *Computational Complexity of One Tape Turing Machine Computations*, J. Assoc. Comp. Mach., vol. 15, 1968, p. 325-339.
13. J. E. HOPCROFT et J. D. ULLMAN, *Formal Languages and their Relations to Automata*, Addison Wesley, 1969.
14. J. KATZENELSON, *The Markov Algorithm as a Language Parser; Linear Bounds*, J. Comp. System Sc., vol. 6, 1972, p. 465-478.
15. M. R. LAGANA, G. LEONI, R. PINZANI et R. SPRUGNOLI, *Improvements in the Exexution of Markov Algorithms*, Bull. Math. Ital., vol. 11, 1975, p. 473-489.
16. G. LEONI et R. SPRUGNOLI, *Some Relations between Markov Algorithms and Formal Languages*, Inst. Sc. Infor. Report, S-76-4, 1976, Pisa.
17. A. A. MARKOV, *Theory of Algorithms*, Traduction anglaise : Israel program for scientific translations, Jérusalem, 1961.
18. G. MICHEL, Thèse de 3e cycle, Université de Rennes, 1975.
19. J. M. MORRIS et V. R. PRATT, *A Linear Pattern Matching Algorithm*, Technical Report n° 40, Univ. of California, Berkeley, 1970.
20. M. PAGET, *Applications des algorithmes de Markov à la complexité des programmes*, Thèse de 3e cycle, Institut de Programmation, Université Paris-VI, 1976.
21. R. W. RITCHIE, *Class of Predictably Computable Functions*, Trans. A.M.S., vol. 106, 1963.
22. A. VAN WIJNGAARDEN, *Recursive Definition of Syntax and Semantics* in Proc. I.F.I.P., 1964, North Holland, Amsterdam, 1966, p. 13-24.

ANNEXE

INSTRUCTIONS DE LA MACHINE A MÉMOIRE ASSOCIATIVE

Instructions de repérage et d'écriture

Instruction	Notation	Effet
Activer	Act α	$\forall i \leqq m$, $\langle R(i, t+1) \rangle := si\ p_1^2(\langle R(i, t) \rangle) = \alpha$ *alors* $(\alpha, 1)$ *sinon* $\langle R(i, t) \rangle$
Désactiver	DACTA α	$\forall i \leqq m$, $\langle R(i, t+1) \rangle := si \langle R(i, t) \rangle = (\alpha, 1)$ *alors* $(\alpha, 0)$ *sinon* $\langle R(i, t) \rangle$
Écrire dans les registres actifs	ECR α	$\forall i \leqq m$, $\langle R(i, t+1) \rangle := si\ p_2^2(\langle R(i, t) \rangle) = 1$ *alors* $(\alpha, 0)$ *sinon* $\langle R(i, t) \rangle$
Mettre actifs tous les registres	MTA	$\forall i \leqq m$, $\langle R(i, t+1) \rangle := (p_1^2(\langle R(i, t) \rangle), 1)$
Mettre inactifs tous les registres	MTI	$\forall i \leqq m$, $\langle R(i, t+1) \rangle := (p_1^2(\langle R(i, t) \rangle), 0)$

Instructions de sélection

Sélecter le registre actif le plus à gauche	SAG	Si $\forall j \leqq m$, $P_2^2(\langle R(j, t) \rangle) = 0$ *alors* $\langle R(j, t+1) \rangle := \langle R(j, t) \rangle$ *sinon* soit $i_0 = \mu i\,(\forall k > i, p_2^2(\langle R(k, t) \rangle = 0)$ $\langle R(j, t+1) \rangle := si\ j = i_0$ *alors* $\langle R(j, t) \rangle$ *sinon* $(p_1^2(\langle R(j, t) \rangle), 0)$
Sélecter le registre actif le plus à droite	SAD	$\forall i \leqq m$, $R(i, t+1) := si\ i = \mu j\,(\forall k > j\ p_2^2(\langle R(k, t) \rangle) = 0$ *et* $p_2^2(\langle R(j, t) \rangle) = 1$ *alors* $(p_1^2(\langle R(j, t) \rangle), 1)$ *sinon* $(p_1^2(\langle R(i, t) \rangle), 0)$

Instructions de décalages

Décalage à droite de l'activité	DAD	$\forall i \leq m$, $\langle R(i+1, t+1) \rangle := si\ p^2_2(\langle R(i,t) \rangle) = 1$ *alors* $(p^2_1(\langle R(i+1,t) \rangle), 1)$ *sinon* $(p^2_1(\langle R(i+1,t) \rangle), 0)$ et $m := m+1$
Décalage à gauche de l'activité	DAG	$\forall i, 0 \leq i \leq m$, $\langle R(i-1, t+1) \rangle := si\ p^2_1(\langle R(i,t) \rangle) = 1$ *alors* $(p^2_1(\langle R(i-1,t) \rangle), 1)$ *sinon* $(p^2_1(\langle R(i-1,t) \rangle), 0)$
Propagation à droite de l'activité	PAD	*Si* $\forall j \leq m$, $p^2_2(\langle R(j,t) \rangle) = 0$ *alors* $\langle R(j,t+1) \rangle := \langle R(j,t) \rangle$ *sinon soit* $i_0 = \mu j \leq m\ (\forall k \geq j\ p^2_2(\langle R(k,t) \rangle) = 0)$ $\forall i < i_0, \langle R(i,t+1) \rangle := \langle R(i,t) \rangle$ $\forall i, i_0 \leq i \leq m \langle R(i,t+1) \rangle := (p^2_1(\langle R(i,t) \rangle), 1)$ et $m := m+1$
Propagation à gauche de l'activité	PAG	*Si* $\forall j \leq m$, $p^2_2(\langle R(j,t) \rangle) = 0$ $\langle R(j,t+1) \rangle := \langle R(j,t) \rangle$ *sinon soit* $i_0 = \mu j (p^2_2 \langle R(j,t) \rangle = 1)$ *alors* $\forall i < i_0, \langle R(i,t+1) \rangle := (p^2_1(\langle R(i,t) \rangle), 1)$ $\forall i, i_0 \leq i \leq m \langle R(i,t+1) \rangle := \langle R(i,t) \rangle$
Décaler les registres actifs à droite	DMAD	$\forall i \leq m$, $\langle R(i+1, t+1) \rangle := si\ p_1(\langle R(i,t) \rangle) = 1$ *alors* $\langle R(i,t) \rangle$ *sinon* $\langle R(i+1,t) \rangle$ *et* $m := m+1$
Décaler les registres actifs à gauche	DMAG	$\forall i, 1 \leq i \leq m$, $\langle R(i-1, t+1) \rangle := si\ p^2_1(\langle R(i,t) \rangle) = 1$ *alors* $\langle R(i,t) \rangle$ *sinon* $\langle R(i-1,t) \rangle$ *et* $m := m-1$

Instructions de branchement

Branchement inconditionnel	B m	branchement inconditionnel à l'instruction d'étiquette m
Branchement si pas de registres actifs	BNA m	branchement à l'instruction d'étiquette m s'il n'existe pas de registres actifs
Branchement si 1 registre actif	B 1 A m	branchement à l'instruction d'étiquette m s'il existe un seul registre actif
Branchement si 2 ou plusieurs registres actifs	B 2 A m	branchement à l'instruction d'étiquette m s'il existe au moins deux registres actifs
Stop	STOP	