Compression de dictionnaires électroniques

Lamia Tounsi, Béatrice Bouchou, Christophe Lenté, Denis Maurel

Université François Rabelais Tours, Laboratoire d'Informatique, France

{lamia.tounsi, beatrice.bouchou, christophe.lente, denis.maurel}@univ-tours.fr

Abstract

Finite state automata are usually used to store electronic dictionaries. This representation allows a quick access to information. However, the size of such automata remains big. Thus, as in a text, when the quantity of information remains large, indexing, ranking and compressing becomes interesting. This work proposes a compression and factorisation algorithm to reduce the memory required to store the automata and to preserve an effective access to data. The main propositions are, on the one hand, the application of the direct acyclic word graph, initially dedicated for indexing text, to index the subautomata, and, on the other hand, heuristic to select the most interesting substructure to factorize. The best candidates to be factorized are those which increase memory storage efficiency and reduce the size of the initial automaton.

Résumé

Les automates à nombre fini d'états sont utilisés entre autres pour le stockage des dictionnaires de langues. L'atout majeur de cette représentation est l'accès rapide à l'information. Cependant, le volume de tels automates reste imposant. Ainsi, comme dans un texte, quand la quantité d'information est suffisamment importante, il devient judicieux de l'indexer, de la classer et/ou de la compresser. Aussi, nous proposons dans cet article un algorithme de factorisation et compression qui permet de réduire l'espace mémoire nécessaire au stockage des automates et de conserver un accès efficace aux données. Nous avons développé diverses versions utilisant, d'une part, l'automate des suffixes (initialement dédié pour l'indexation de texte) pour indexer les sous-automates, et, d'autre part, des heuristiques permettant de sélectionner les sous-structures les plus intéressantes à factoriser : celles qui maximisent le gain de mémoire et réduisent la taille de l'automate initial.

Mots-clés : automate, sous-automate, indexation, DAWG, compression, factorisation, dictionnaire, algorithme glouton.

1. Introduction

Les dictionnaires électroniques des langues naturelles sont essentiels pour l'analyse et la génération automatique de textes. La représentation des dictionnaires par des automates acycliques est très répandue, elle permet de compacter les données tout en préservant un accès direct à l'information (Revuz, 1991). Plusieurs études proposent des variantes de compression de ces gros volumes de données (Boubaker, 1996 ; Daciuk, 2000 ; Ristov et Laporte, 1999 ; Ristov, 2005) et c'est dans ce cadre que se situe notre travail.

Cet article décrit une palette d'outils pour compresser et indexer des automates représentant des dictionnaires. Ainsi, nous proposons un algorithme de compression qui permet, d'une part, de réduire l'espace mémoire nécessaire au stockage des automates et, d'autre part, de conserver un accès efficace aux données. Par extrapolation de l'indexation automatique d'un document texte qui recense toutes les occurrences des mots de ce document, l'indexation automatique d'un automate liste tous ses sous-automates accompagnés de leurs adresses (positions dans l'automate) pour former son index. Ainsi, les sous-automates constituent une

information accessible et représentative du contenu de l'automate. Étant donné que la recherche de sous-automates a révélé la présence massive de sous-automates de type série ou parallèle (Tounsi et al., 2006), nous nous sommes intéressés spécifiquement à ces deux types de sous-structures et à leur indexation et compression. Pour indexer ces parties internes d'un automate, nous avons étudié la possibilité d'utiliser des systèmes dédiés initialement à l'indexation des textes. En l'adaptant à nos besoins, l'automate des suffixes (Directed Acyclic Word Graph « DAWG ») permet d'avoir accès, non seulement aux positions des séries et des parallèles dans l'automate, mais aussi, à l'ensemble des positions des sous-séries et des sous-parallèles dans ce même automate. Pour alléger l'automate, nous proposons un algorithme glouton qui factorise l'automate. Sans la possibilité d'un retour arrière, il sélectionne à chaque étape le sous-automate qui apporte le meilleur gain avant de le factoriser. Ensuite, cet algorithme a été mis en œuvre pour traiter plusieurs stratégies de compression possibles.

Cet article se présente comme suit : tout d'abord, nous introduisons les définitions des automates et des sous-structures détectées dans les automates, ensuite nous présentons notre adaptation du DAWG, puis l'algorithme de factorisation-compression.

2. Définitions, notations et abréviations

2.1. Automates et sous-automates

Soit $A=< \Sigma, Q, \delta, q_i, q_f >$ un *automate déterministe minimal acyclique à nombre fini d'états* où : Σ est l'alphabet, Q est l'ensemble fini d'états, δ est une fonction de transition $\delta : Q \times \Sigma \rightarrow Q$, q_i est l'état initial et q_f est l'état final. Les ensembles de successeurs et de prédécesseurs d'un état $p \in Q$ sont toujours calculés dans l'automate A comme suit :

$$Succ(p)=\{q\in Q : \exists\, \alpha\in\Sigma : \delta(p, \alpha) =q\} \text{ et } Succ^*(p) = \{q\in Q : \exists\, w \in\Sigma^* : \underline{\delta}(p,w)=q\}$$

$$Pred(p)=\{q\in Q : \exists\, \alpha\in\Sigma : \delta(q, \alpha) =p\} \text{ et } Pred^*(p) = \{q\in Q : \exists\, w \in\Sigma^* : \underline{\delta}(q,w)=p\}$$

Succ(p) (resp. Pred(p)) est l'ensemble des successeurs (resp. prédécesseurs) immédiats de p.

Succ*(p) (resp. Pred*(p)) est l'ensemble de tous les successeurs (resp. prédécesseurs) de p, incluant l'état p.

Étant donné que nous souhaitons rechercher des structures isolées qui peuvent être extraites de l'automate et remplacées par une transition, nous avons retenu la définition suivante d'un sous-automate (Tounsi 2007).

Soit $A=< \Sigma, Q, \delta, q_i, q_f >$ un automate acyclique à nombre fini d'états, $A' =< \Sigma, Q', \delta', s_i, s_f >$ est un *sous-automate* de A si et seulement si :

- $Q' \subset Q$, $\{s_i, s_f\} \subset Q'$,
- δ' :
  - $Q' \times \Sigma \rightarrow Q'$
  - $\forall (q, \alpha) \in Q' \setminus \{s_i, s_f\} \times \Sigma$: Si $\delta(q, \alpha)$ est défini alors $\delta'(q, \alpha) = \delta(q, \alpha)$ sinon $\delta'(q, \alpha)$ est indéfini
  - $\forall \alpha \in \Sigma$, si $\delta(s_i, \alpha) \in Q'$ alors $\delta'(s_i, \alpha) = \delta(s_i, \alpha)$ sinon $\delta'(s_i, \alpha)$ est indéfini
  - $\forall \alpha \in \Sigma$, $\delta'(s_f, \alpha)$ est indéfini
- $\forall q \in Q'$, $q \in Succ^*(s_i)$ et $q \in Pred^*(s_f)$
- $\forall q \in Q' \setminus \{s_i, s_f\}$: $Succ(q) \subset Q'$ et $Pred(q) \subset Q'$.

La recherche des sous-automates a révélé la présence massive de sous-automates de type série ou parallèle, raison pour laquelle nous nous intéressons spécifiquement à ces deux types de sous-structures et à leur indexation.

2.2. Représentation et stockage d'un automate

Le travail présenté dans cet article s'applique à des automates acycliques avec un seul état final (Maurel et Guenthner, 2006).

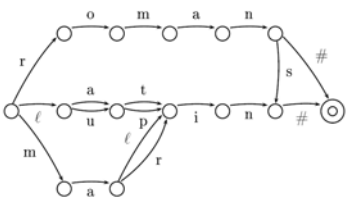
En mémoire, nous représentons l'automate par la liste des transitions sortantes de chaque état. Chacune de ces transitions porte trois informations : *un booléen* mis à zéro ou à 1 selon que la transition partage ou pas le même état source que celle qui la précède dans la liste, son *étiquette* et l'adresse de la première transition de *son état but* (cf. Exemple donné figure 1).

2.3. Représentation étendue et stockage d'un automate

En s'appuyant sur la méthode présentée dans (Tounsi et al., 2006), on peut remplacer itérativement les sous-automates séries et les sous-automates parallèles par de simples transitions portant un nouveau label jusqu'à ce que l'automate en soit dénué. Cela revient en fait à étendre l'alphabet avec les nouveaux labels (voir figure 1). Pour cela on numérote à partir de 1 l'alphabet initial et on associe les numéros au delà de la dernière lettre aux sous-structures traitées. Le stockage se décompose en deux parties, le fichier habituel décrivant un automate auquel s'ajoute un fichier répertoriant l'alphabet étendu. Une série est représentée par la concaténation des étiquettes de ses transitions dans leur ordre d'apparition. Un parallèle est lui aussi représenté par la concaténation des étiquettes de ses transitions selon un ordre pré-établi sur l'alphabet. De plus, il est associé à chaque mot un booléen précisant la nature de la sous-structure qu'il représente, il prend la valeur vrai pour désigner une série et faux pour désigner un parallèle, ce qui est noté respectivement S et P sur la figure.

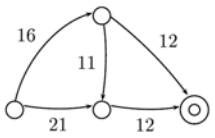
Automate initial A

1	1	ℓ	7
2	0	m	6
3	0	r	4
4	1	o	5
5	1	m	9
6	1	a	10
7	1	a	12
8	0	u	12
9	1	a	14
10	1	ℓ	15
11	0	r	15
12	1	p	15
13	0	t	15
14	1	n	17
15	1	i	16
16	1	n	19
17	1	#	0
18	0	s	19
19	1	#	0



Représentation étendue de A

1	1	16	3
2	0	21	5
3	1	11	5
4	0	12	0
5	1	12	0



Alphabet		
Alphabet initial		
1		ℓ
2		m
3		r
4		o
5		a
6		u
7		p
8		t
9		n
10		i
11		s
12		#
Alphabet étendu		
13	P	5 6
14	P	7 8
15	P	1 3
16	S	3 4 2 5 9
17	S	1 13 14
18	S	2 5 15
19	S	10 9
20	P	17 18
21	S	20 19

Figure 1 : Représentation étendue d'un automate

2.4. Optimisation de l'occupation mémoire d'un automate

Les automates sont souvent stockés dans des fichiers textes. Dans le but de réduire l'espace mémoire nécessaire, nous avons codé chaque donnée du même type sur un nombre de bits minimal, mais constant. Cette méthode permet, d'une part, de stocker les automates à moindre coût et, d'autre part, de conserver un accès efficace aux données. Le codage réservé à chaque

donnée est le suivant : i) Codage du booléen : 1 bit. ii) Codage des étiquettes : le nombre de bits nécessaires pour coder l'alphabet, noté $Taille_{alphabet}$: $Taille_{alphabet} = \lceil \log_2 (|\Sigma'|) \rceil$ où Σ' est l'union de l'alphabet initial Σ et des nouveaux caractères alloués aux séries et parallèles détectés et factorisés. iii) Codage de l'adresse de l'état d'arrivée : le nombre de bits nécessaires pour coder les adresses, noté $Taille_{adresse}$: $Taille_{adresse} = \lceil \log_2 (ValMax+1) \rceil$ où ValMax, correspondant à l'adresse de l'état but de valeur maximale.

L'adresse est une adresse absolue puisque le dictionnaire contient des mots courts, ce qui nécessite des grands sauts d'adresse et qui rend donc inutile un adressage relatif.

Exemple 1 : La figure 1 présente un automate initial et sa représentation en mémoire ainsi que sa représentation étendue où il est composé de 4 états et 5 transitions. À cet automate est associé un alphabet étendu contenant l'alphabet initial, 12 caractères et 9 sous-structures distribuées en 4 parallèles et 5 séries. Ainsi, le « caractère » 13 représente le parallèle composé des transitions a et u. Dans cet exemple, 10 bits sont nécessaires pour coder l'automate initial car : $Taille_{alphabet} = 4$ et $Taille_{adresse} = 5$.

3. Indexation des séries et parallèles

Pour indexer les séries, une liste est réalisée associant à chaque série l'ensemble de toutes ses positions dans l'automate. La position d'une série dans l'automate est donnée par le numéro de ligne de sa transition de départ dans le fichier représentant l'automate. Une liste similaire est dressée pour indexer les parallèles. On constate qu'il existe des structures qui sont inclues dans d'autres structure et sur plusieurs niveau d'imbrication. Ainsi une série peut être une sous-série d'une autre série plus longue.

3.1. Indexation des sous-séries et sous-parallèles

Pour identifier les imbrications de structure, nous avons étudié la possibilité d'utiliser des systèmes dédiés initialement à l'indexation des textes pour avoir accès, non seulement aux positions des séries et des parallèles, mais aussi à l'ensemble des positions des sous-séries et des sous-parallèles dans l'automate. Une technique, qui répond à nos besoins, est l'automate des suffixes « DAWG ». En effet, plusieurs études autour de l'indexation des répétitions de structures, des mots et des textes, font usage d'un DAWG (Blumer et al 1985 ; Crochemore et al, 1997 ; Crochemore et al 2001). Dans notre cas, l'utilisation d'un DAWG nous permet de calculer les positions de toutes les occurrences d'une sous-structure, en y détectant des inclusions, des chevauchements, etc.

3.2. Automate des suffixes « DAWG »

Un DAWG est un automate des suffixes qui représente une structure efficace pour le traitement et l'analyse des répétitions dans un texte. Son algorithme de construction repose sur une lecture de gauche à droite du texte pour créer au fur et à mesure l'automate minimal des suffixes en temps linéaire. Il représente ainsi le texte sous forme d'un graphe et à chaque état est associée l'ensemble des positions des mots reconnus, à cet état, dans le texte.

Comment adapter le DAWG ?

Nous avons adapté ce procédé à notre contexte dans le but d'indexer et de comparer les sous-structures détectées dans les automates (séries ou parallèles[1]). Les étapes ci-dessous décrivent l'adaptation aux séries, pour les parallèles les mêmes étapes s'appliquent, avec un traitement supplémentaire.

- Transformation des données d'entrée : Les séries sont converties en mots pour s'adapter au format de données manipulé par le DAWG. Un mot représentant une série est formé par la concaténation des étiquettes des transitions de cette série dans l'ordre de leur apparition.
- De l'intérêt des états finaux : À chaque état d'un DAWG, qu'il soit final ou non, correspond au moins une sous-série qui peut être préfixe, infixe ou suffixe d'une série de la liste initiale.
- Utilisation d'un index double dans le DAWG : À chaque état d'un DAWG correspond un index qui indique les positions (dans l'automate initial) des sous-séries associées à l'état. Cet index comporte deux parties : i) la liste des séries contenant ces sous-séries, ii) les positions des sous-séries dans les séries.

Traitement des parallèles :

Soit S une série de l'automate et P un parallèle, une sous-série de S est forcément formée par la concaténation d'une suite d'étiquettes des transitions S, dans l'ordre de leur apparition. Un sous-parallèle de P est formé par la concaténation des étiquettes des transitions P mais pas forcément dans l'ordre de leur apparition.

Comme pour les séries, les parallèles sont transformés en mots. Un mot représentant un parallèle est formé par la concaténation ordonnée des étiquettes par exemple suivant l'ordre alphabétique, permettant ainsi de détecter les parallèles identiques. Néanmoins, un traitement supplémentaire est nécessaire car le DAWG associé au mot représentant un parallèle ne permet pas de produire tous ses sous-parallèles. Pour les créer, il faut associer plusieurs mots à un parallèle. Plus exactement $2^{(n-2)}$ mots, où n représente le nombre de transitions du parallèle. En général, n est suffisamment petit pour que ce ne soit pas pénalisant.

Par exemple, pour générer tous les sous-parallèles du parallèle « abcde », on va créer le DAWG associé aux mots : « ae », « abe », « ace », « abce », « ade », « abde », « acde », « abcde ».

En l'adaptant ainsi, le DAWG constitue un outil bien adapté à l'indexation des séries et des parallèles.

4. Factorisation et compression des automates

L'étape de factorisation est combinée à l'indexation et se situe en amont de la compression. En effet, la factorisation permet de réorganiser l'automate dans le but de réduire la quantité de données nécessaires pour le représenter. Ainsi, l'objectif est d'alléger l'automate en diminuant son nombre d'états et son nombre de transitions sous la condition stricte de ne pas modifier le langage initialement reconnu.

[1] Pour alléger la recherche et faciliter la lecture des informations de l'index d'un DAWG, les séries et les parallèles sont traités séparément.

Les résultats de la factorisation totale des automates étudiés montrent que toutes les sous-structures ne sont pas forcément intéressantes à factoriser. Effectivement, lorsqu'on factorise une série ou un parallèle non redondant on augmente forcément la taille de l'alphabet en la remplaçant dans l'automate par une transition lui faisant référence. Pour éviter d'augmenter la taille de stockage, il est indispensable de choisir les bons éléments à factoriser.

4.1. Comment choisir les sous-structures à factoriser ?

Pour choisir les sous-structures à factoriser, nous utilisons une fonction appelée « fonction gain ». Elle calcule pour chaque sous-structure, d'une part, l'espace mémoire économisé en supprimant toutes ses occurrences de l'automate et, d'autre part, l'espace mémoire consommé en l'insérant à l'alphabet. La différence indique s'il est avantageux de factoriser cette sous-structure ou non.

Gain= Bénéfice - Préjudice

Le bénéfice est mesuré en calculant le nombre de bits libérés par la suppression de la sous-structure et l'ensemble de ses occurrences de l'automate, plus précisément, il s'agit du nombre de bits permettant de stocker chaque transition de la sous-structure, à chacune de ses positions, dans l'automate initial.

Le préjudice est subi par le nombre de bits nécessaires au stockage de la sous-structure en mémoire et à l'extension de l'alphabet. Plus précisément, il s'agit de comptabiliser le nombre de bits utilisés par chaque transition qui remplace la sous-structure dans l'automate initial, auquel s'ajoute, le nombre de bits utilisés pour stocker le mot représentant la sous-structure et son lien avec le nouveau caractère de l'alphabet.

Les paramètres de la fonction gain sont déterminés à partir du codage du fichier compressé. Étant donné que les tailles calculées varient au fur et à mesure de la factorisation, il est nécessaire d'émettre plusieurs hypothèses, dès le départ, pour fixer ces tailles. Les hypothèses vont porter principalement sur les paramètres $\text{Taille}_{\text{alphabet}}$ et $\text{Taille}_{\text{adresse}}$.

4.1.1. Calcul des fréquences à l'aide d'un DAWG

Déjà utilisé à l'étape de l'indexation, le DAWG fournit également ici l'information sur les fréquences. En effet, les index associés à chaque état représentent les occurrences des sous-structures.

4.1.2. Comment factoriser ?

Une sous-structure peut être incluse dans une autre sous-structure, aussi, on ne peut pas prévoir l'impact d'une factorisation, ou d'une combinaison de factorisations, sur les autres. **Exemple 2 :** Soient S_1, S_2 et S_3 trois séries détectées dans un automate A : S_1= ccdbaab, S_2= baab et S_3= ccdb.

Supposons que S_1 apparaisse quatre fois dans A et que, en dehors de S_1, S_2 apparaisse une fois dans A et S_3 y apparaisse deux fois. Donc au total, S_2 apparaît cinq fois dans A et S_3 six fois.

Supposons enfin que l'alphabet étendu contient au maximum 8 caractères et que l'automate contient moins de 128 transitions ; on a donc besoin en mémoire de 11bits pour stocker une transition (1 bit pour le booléen, 3 bits pour l'étiquette et 7 bits pour le numéro de la transition d'arrivée). Lorsque l'on factorise une sous-structure de L transitions et de fréquence F, on gagne F*(L-1)*11 bits (L-1 car on remplace la sous-structure par une transition dans l'automate) : Bénéfice = F*(L-1)*11 bits.

Pour indexer la sous-structure, on l'ajoute à la liste de l'alphabet étendu en indiquant sa taille (le nombre de ses étiquettes) et la liste de ses étiquettes. Dans cet exemple, la taille d'une sous-structure est forcément inférieure à 7, 3 bits sont donc nécessaires pour la stocker et la liste des étiquettes demande 3*L bits. Ce qui fait au total 3+3L bits : Préjudice = 3+3L bits.

$$\text{Gain} = F * (L - 1) * 11 - 3 - 3L \text{ bits}$$

La figure 2 présente le DAWG construit pour S_1, S_2 et S_3. A chaque état du DAWG est associé un index représentant les positions des sous-séries qu'il reconnaît dans S_1, S_2 et S_3. Cet index indique aussi la fréquence des sous-séries reconnues à cet état.

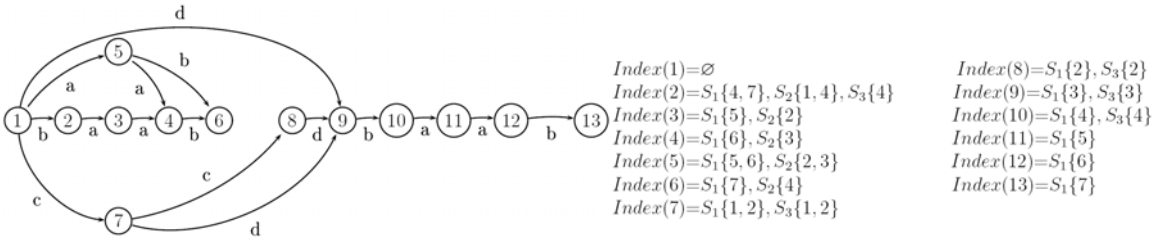


Figure 2 : DAWG (S1, S2, S3)

- Si on factorise 4 S_1 puis 2 S_3 : Gain = 291 bits.

- Si on factorise 5 S_2 puis 6 ccd : Gain = 292 bits.

- Si on factorise 5 S_2 puis 4 ccd et ensuite 2 S_3 : Gain = 277 bits.

- Si on factorise 5 aab puis 6 S_3 : Gain = 183 bits.

Plusieurs scénarios de factorisation sont possibles et chacun engendre un gain différent. En effet, à chaque état p du DAWG correspond une ou plusieurs sous-structures et à chaque sous-structure est associé son gain (calculé en fonction de sa longueur et de sa fréquence).

Puisque le but est bien évidemment d'alléger le traitement, nous proposons une première heuristique pour limiter l'intervalle des valeurs à explorer et favoriser la factorisation des plus longues séries. Ainsi, il sera associé à chaque état p le gain de la factorisation de la plus longue de ses sous-structures. La profondeur de l'état p (la taille du plus long chemin reliant l'état initial du DAWG à p) indique la taille de la plus longue sous-série reconnue à p.

4.1.3. Comment adapter le DAWG à la factorisation ?

La factorisation d'une sous-structure w_i implique des mises à jour de l'automate initial mais aussi du DAWG(w_1,..,w_n). En effet, lorsqu'elle est choisie, w_i est retirée du DAWG(w_1,..,w_n) par la suppression de tous les index lui faisant référence. Lorsqu'un état du DAWG(w_1,..,w_n) possède un index vide, il est lui aussi éliminé.

Si $\exists\ w_j \in \{w_1,..,w_n\}$: $w_i \subset w_j$ et $|w_i| < |w_j|$ alors la factorisation de w_i crée une sous-structure w_{n+1} en remplaçant w_i par la nouvelle lettre de l'alphabet lui faisant référence dans w_j.

Le DAWG évolue au fur et à mesure de cette extension et intègre aisément tout nouvel élément.

4.2. Algorithme glouton de factorisation

Nous proposons un algorithme glouton pour factoriser l'automate. Le principe général de cet algorithme est de déterminer à chaque étape, le meilleur choix local (optimum local), sans aucun retour arrière, en espérant obtenir la bonne factorisation globale. Il localise ainsi, au fur

et à mesure, la sous-structure la plus intéressante à factoriser à l'instant t. Les étapes ci-dessous décrivent le principe de cet algorithme pour les séries. Pour les parallèles les mêmes principes s'appliquent.

Notre méthode de factorisation est basée sur la construction et la réduction d'un DAWG qui indexe toutes les séries localisées dans un automate donné en entrée A. Elle passe par les étapes suivantes :

- Examiner les états du DAWG pour déterminer la sous-série la plus intéressante à factoriser : celle qui produit le meilleur taux de compression à l'instant t (optimum local).

- Factoriser la sous-série dans A. L'index du DAWG permet de localiser toutes les positions de la sous-série dans l'automate.

- Supprimer la sous-série du DAWG en éliminant tous les index lui faisant référence, ensuite supprimer les états qui possèdent un index vide.

- Mettre à jour le DAWG en y insérant les nouvelles séries : celles produites par la substitution de la sous-série dans toutes les séries où elle apparaît. La nouvelle série est représentée par un nouveau caractère de l'alphabet.

Ce processus est itératif et il s'achève lorsqu'il n'existe plus de sous-série intéressante à factoriser ou lorsque la taille maximale de l'alphabet est atteinte.

ALGORITHME : FACTORISATION ET COMPRESSION **ENTRÉE/SORTIE : A - SORTIE : CSA**
Soit SA l'ensemble des séries : SA←∅ ;
Soit CSA l'ensemble des séries factorisées : CSA←∅ ;
Calculer l'ensemble SA à partir de l'automate A ;
Créer DAWG (SA) et calculer le gain pour chacun de ses états ; */*Utiliser la fonction gain*/*
Sélectionner l'état q qui possède le meilleur gain à partir du DAWG (SA) ;
TANT QUE DAWG (SA) contient au moins un état q possédant un gain attractif
  Soit w le plus long chemin arrivant à q et α un nouveau caractère ; */*alphabet étendu*/*
    Supprimer w du DAWG (SA) ;
    Remplacer w par une transition étiquetée par α dans A ;
    Remplacer w par une transition étiquetée par α dans SA ;
    Ajouter la série w à CSA ;
    Mise à jour du DAWG (SA) avec les nouveaux éléments de SA ;
    Sélectionner l'état q qui possède le meilleur gain à partir du DAWG (SA) ;
FIN TANT QUE

5. Résultats expérimentaux

5.1. Les dictionnaires et automates étudiés

Dans cette partie, nous présentons les dictionnaires étudiés et leurs automates associés ainsi que quelques statistiques générales. Nous distinguons deux catégories de dictionnaires :

Catégorie 1 : Ce sont des dictionnaires électroniques construits selon le formalisme « DELA[2] » (dictionnaires électroniques du LADL).

[2] Ce modèle réunit un ensemble de mots et associe à chaque mot une série d'informations morphologiques, syntaxiques et sémantiques (B. Courtois, 1990, 1999). Toutefois, dans la suite du travail, il sera tenu compte uniquement de l'ensemble des mots présents dans un dictionnaire et non des informations associées

Catégorie 2 : Ce sont des dictionnaires électroniques élaborés par une équipe de recherche de l'Université de Neuchâtel. Ces dictionnaires représentent une collection de mots simples et de mots composés collectés sur des sites web et essentiellement à partir d'éditions électroniques de journaux. La particularité de ces dictionnaires est de contenir des chiffres et plusieurs caractères spéciaux tels que le l'arobase, l'esperluette, le point, etc.

Le tableau 1 présente les caractéristiques globales des dictionnaires et automates examinés, le nombre de mots, la taille de l'alphabet, le nombre de transitions, le nombre d'états, la taille de l'automate. On remarque que les automates représentant ces dictionnaires sont très grands et ce, malgré leur minimisation. A titre d'exemple, le dictionnaire DELAF Fr contient 637 282 mots et son alphabet initial se compose de 71 caractères. L'automate initial qui le représente contient 177 465 transitions et 67 995 états et sa taille est 2 178 Ko. Cet automate contient 14 624 séries et 1 706 parallèles. Après la factorisation de ces sous-structures, cet automate ne contient plus que 150 299 transitions et 42 625 états et sa taille devient 563 Ko.

	Dictionnaire		Automate initial			Représentation étendu de l'automate				
	Nb mots	Alphabet	Tr	états	Taille	Tr	états	Série	Parallèle	Taille
Catégorie 1										
DELAF Fr	637 282	71	177 465	67 995	2 178	150 299	42 625	14 624	1 706	563
DELAF En	282 338	74	252 664	116 848	3 081	199 332	66 635	26 132	2 871	801
DELAF Sr	1 214 417	52	193 668	61 383	2 340	163 466	43 018	11 106	8 372	591
DELAF De	3 713 121	89	335 284	142 795	4 133	276 223	87 772	292 599	3 384	1 105
Villes Fr	35 391	75	95 589	61 240	1 108	56 064	22 224	12 754	448	291
Poly En	320 424	31	717 112	435 940	8 963	427 269	147 423	74 888	1 264	2 451
Catégorie 2										
Web Fr	236 057	43	298 117	101 837	3 560	253 595	72 102	17 110	10 235	946
Web Hu	191 738	43	270 495	113 678	2 503	221 908	72 006	23 811	5 112	638
Web Bg	165 073	42	209 209	85 661	3 235	167 045	52 052	18 069	5 792	858
Web Pt	398 839	49	538 697	214 992	6 510	435 147	130 430	38 392	13 270	1 776

Tableau 1 : Dictionnaires et automates

5.2. Résultats de la Compression

Nous avons testé plusieurs variantes de factorisation et compression. Pour chacune, nous nous sommes limités à la factorisation des sous-automates séries étant donné que les sous-automates parallèles ne représentent en moyenne que 1% des transitions de l'automate. Cependant le processus est exactement le même pour traiter les parallèles.

Trois variantes de factorisation ont été testées (utilisant l'algorithme factorisation-compression). La première s'applique directement sur des automates minimisés (FCM), la deuxième considère les automates non minimisés (FCNM) et la troisième traite directement du texte et s'applique aux dictionnaires avant même de générer leur automate associé (FCDic). Pour pousser plus loin l'expérimentation sans remettre en question les différentes

approches, nous avons considéré les données à l'envers en inversant les mots des dictionnaires. Ainsi, les trois variantes de factorisation ont été, à nouveau, testées sur des ensembles d'entrée métamorphosés.

Le tableau 2 présente un résumé des meilleurs résultats de compression. A titre d'exemple, la taille initiale du fichier texte contenant l'automate DELAF Fr est 2 178 Ko, lorsque qu'il est stocké dans un fichier binaire en associant l'espace nécessaire et suffisant pour coder chaque information, sa taille est 563 Ko. La meilleure compression est obtenue en par la méthode FCM qui réduit sa taille à 529 Ko avec un alphabet étendue à 128 caractères.

On constate qu'en général, les factorisations les plus efficaces des automates se réalisent en utilisant un alphabet assez court, de 7 à 8 bits. Toutefois, on remarque qu'il existe un intrus dans ce tableau, il s'agit de l'automate des polylexicaux anglais qui atteint son meilleur taux de compression lorsque la taille maximale de l'alphabet est fixée à 2 048. Ce dictionnaire est composé d'expressions anglaises où nombreux mots y sont récurrents.

Pour résumer, pour chacun des automates, il existe au moins une méthode qui permet de réduire sa taille initial, par contre, il n'existe pas de méthode efficace pour tous les automates. On rejoint ainsi le travail effectué par Daciuk, dans (Daciuk, 2000), qui affirme qu'il n'existe pas de méthode de compression optimale pour tous les automates.

On constate que la méthode FCM est efficace lorsqu'elle s'applique aux automates du DELAF et la méthode FCNM est efficace lorsqu'elle s'applique aux automates de la catégorie 2. Ainsi, compresser un automate non minimal peut être efficace, on rejoint donc le travail de Ristov et Laporte dans (Ristov 1999, Ristov 2005) qui applique une compression LZW à des automates non minimaux pour obtenir de bons taux de compression.

Les méthodes FCM et FCNM proposent des résultats intéressants. Il serait donc justifié d'étudier la possibilité de les combiner.

On constate également que la méthode FCDic profite à certains dictionnaires. En effet, la taille de l'automate représentant le dictionnaire web portugais a baissé jusqu'à atteindre 1 600 Ko. Plus important encore, la taille de l'automate des villes françaises passe à 67 Ko, une réduction de 77% par rapport à sa taille d'origine avec un alphabet étendu à 128 caractères. L'examen des sous-mots factorisés explique ce résultat par la présence de nombreuses sous-structures communes dans les noms de villes françaises telles que, Saint, Bourg, etc.

On remarque également qu'inverser les mots est profitable à l'automate des polylexicaux anglais qui réduit sa taille de 30%.

Automate	Taille initiale fichier texte Ko	Taille initiale codage binaire Ko	Meilleure compression Ko	Méthode utilisée	Taille Alphabet
Catégorie 1					
DELAF Fr	2 178	563	529	FCM	128
DELAF En	3 081	801	735	FCM	256
DELAF Sr	2 340	591	585	FCM	128
DELAF De	4 133	1 105	1 036	FCM	256
Villes Fr	1 108	291	67	FCDic	128
Poly En	8 963	2 451	1 709	$FCM_{inverse}$	2 048
Catégorie 2					
Web Fr	3 560	946	934	FCNM	128
Web Bg	2 503	638	608	FCNM	128
Web Hu	3 235	858	757	FCM	128
Web Pt	6 510	1 776	1 600	FCDic	128

Tableau 2 : Meilleures compressions

5.3. Discussion

Les résultats obtenus correspondant aux différentes variantes testées confirment l'importance du choix des sous-structures à factoriser. Ce choix dépend du nombre de transitions que contiennent ces sous-structures et de leur fréquence dans l'automate, mais il dépend aussi de l'impact d'une factorisation sur les suivantes. La fonction gain associée à chaque état du DAWG permet de calculer le gain de la factorisation de la plus longue série reconnue à cet état, même si cet état reconnaît d'autres séries plus courtes. Il est donc possible d'envisager de ne pas factoriser la plus longue séquence reconnue à un état. Il est aussi possible de factoriser uniquement une partie des occurrences d'une série dans l'automate pour anticiper l'impact des combinaisons de factorisation. Ces deux remarques ouvrent la réflexion sur la possibilité de sélectionner différentes factorisations lors d'une même étape.

Parmi les travaux existants traitant le problème de la compression d'automates représentant des dictionnaires de langues, on peut citer Ristov et Laporte [Ristov1999]. Pour représenter et stocker les données, ils n'utilisent pas des automates finis acycliques, mais des arbres lexicographiques. Une fois généré, un arbre est transformé en une liste chaînée avant d'être compressé suivant la méthode de Ziv et Lempel. La recherche des structures similaires dans l'arbre lexicographique s'adapte aux données et utilise des arbres de suffixes ou des tableaux de suffixes pour y détecter les répétitions. En comparaison au travail proposé dans cet article, le taux de compression du DELAF français obtenu par Ristov et Laporte est meilleur, cependant leur structure de donnée évolue et ne correspond plus à un automate en fin de parcours. Notre structure de données n'est pas modifiée : on utilise toujours des automates finis acycliques pour représenter les dictionnaires.

6. Conclusion

Dans cet article, nous avons présenté une méthode itérative pour indexer, factoriser et compresser les automates acycliques à nombre finis d'états représentant des dictionnaires électroniques. Étant donné un automate A, cette méthode utilise la recherche des sous-automates séries-parallèles pour localiser les sous-structures de A et ensuite calcule, à chaque étape, les gains d'un ensemble de factorisations possibles en utilisant un DAWG.

Nous avons proposé un algorithme glouton qui détermine le meilleur choix local pour tenter d'obtenir la meilleure factorisation globale de A. Il localise, au fur et à mesure, la sous-structure la plus intéressante à factoriser, à partir d'un DAWG qui indexe toutes les sous-structures de A. Une fois choisie, cette sous-structure est remplacée dans A par une extension de son alphabet. Ce processus est itératif et il s'achève lorsqu'il n'existe plus de sous-structure intéressante à factoriser ou lorsque la taille maximale de l'alphabet est atteinte. Les mises à jours de l'automate factorisé A sont possibles, il demeure ainsi toujours utilisable.

Remerciements

Les auteurs remercient le professeur Franz Guenthner, le professeur Éric Laporte, le professeur Jacques Savoy et le professeur Dusko Vitas pour la libre utilisation des différents dictionnaires.

Références

Blumer A., Blumer J., Haussler D., Ehrenfeucht A., Chen M.T. and Seiferas J. (1985). The smallest automaton recognizing the subwords of a text. *Theoret.Comput. Sci*., 40: 31-55.

Crochemore M. and Hancart C. (1997). Automata for matching patters. *Handbook Formal Languages.* Springer.

Crochemore M., Hancart C. and Lecroq T. (2001). *Algorithmique du texte.* Vuibert.

Daciuk J. (2000). Experiments with automata compression. In *Proc. of CIAA'00,* 105-112.

Boubaker M. H. (1996). *Méthodes et algorithme de représentation et de compression de grands dictionnaires de formes*. Thèse de doctorat, Université Josef Fournier, Grenoble 1.

Maurel D. and Guenthner F. (2006). *Automata and Dictionnaries*. King's College Publications.

Revuz D. (1991). *Dictionnaires et lexiques : méthodes et algorithmes*. Thèse de doctorat, Université Paris7.

Ristov S. (2005). LZ trie and dictionnary compression. *Software : Practice and experience* 35(5), 445-465.

Ristov S. and Laporte E. (2000). Ziv Lempel compression of huge natural langage data tries using suffix arrays. In *Combinatorial Pattern Matching, 10th annual Symposium Warwick University, Proceedings,* M. Chrochemore, M. Paterson, LNCS 1645: 196-211.

Tounsi L. (2007). *Sous-automates à nombre fini d'états. Application à la compression de dictionnaires électroniques.* Thèse de doctorat, Université François Rabelais de Tours.

Tounsi L., Lenté C. and Maurel D. (2006). Analyse statistique de la structure des automates représentant des dictionnaires électroniques. In *Proc. of JADT'06*, pages 527-935.