Bibliothèques d'automates finis et grammaires context-free : de nouveaux traitements informatiques

Matthieu Constant
Laboratoire d'Automatique Documentaire et Linguistique (LADL)
Université de Marne-la-Vallée
Batiment Copernic
Champs-sur-Marne
77 457 Marne-la-Vallée
mconstant@univ-mlv.fr

Résumé – Abstract

La quantité de documents disponibles via Internet explose. Cette situation nous incite à rechercher de nouveaux outils de localisation d'information dans des documents et, en particulier, à nous pencher sur l'algorithmique des grammaires context-free appliquée à des familles de graphes d'automates finis (strictement finis ou à cycles). Nous envisageons une nouvelle représentation et de nouveaux traitements informatiques sur ces grammaires, afin d'assurer un accès rapide aux données et un stockage peu coûteux en mémoire.

The amount of documents available over the Internet is exploding. This phenomenon requires the development of new electronic representations and tools to search information into these documents. This article deals with context-free algorithms applied to finite state graphs (strictly finite or with cycles). It shows new methods and representations to combine efficiently complexities in terms of memory space and processing time.

Mots-clés – Keywords

Automates finis, forme normale de Greibach, grammaires context-free, graphes

Context-Free Grammars, Finite State Automata, Graphs, Greibach Normal Form

1 Introduction

Le LADL a entrepris, dès les années 60, une couverture complète du lexique et de la syntaxe de la langue française. L'équipe de recherche a emmagasiné trois types de ressources linguistiques : tables de lexique-grammaire, dictionnaires électroniques et grammaires locales (C. Leclère et al. 1991). Les informaticiens du LADL ont mis en place de nouvelles représentations à états finis, à la fois, claires, à accès rapide et peu coûteuses en mémoire. Les dictionnaires sont stockés sous la forme d'automates de caractères (D. Revuz 1991). Les grammaires locales ont la forme de graphes d'automates finis. A terme, les tables de lexique-

grammaire, ressources fondamentales pour l'analyse syntaxique automatique, devraient être transformées de manière systématique en graphes d'automates finis (E. Roche 1993). Le logiciel INTEX (M. Silberztein 1993) permet aux linguistes d'acquérir, de créer et d'appliquer ces ressources d'une manière claire et simple. Ainsi, nous prévoyons, dans un avenir proche, une forte augmentation du nombre de graphes. Ces graphes sont transformés en transducteurs à états finis (FST) afin de les appliquer à des textes. Cette opération augmente la place en mémoire nécessaire, de manière exponentielle. L'algorithmique des automates finis présente des avantages : déterminisation et minimisation sont a priori de gros atouts. Toutefois, ces outils classiques montrent leurs limites. Ainsi, la minimalisation, trop coûteuse en temps, a été abandonnée et la déterminisation, bien qu'indispensable à l'amélioration des temps d'accès, a tendance à faire exploser la mémoire. Cette perspective nous incite à nous pencher sur l'algorithmique des grammaires context-free. Dans un premier temps, nous décrivons le système actuel et ses limites. Puis, nous montrons une nouvelle représentation des grammaires locales et les traitements associés. Enfin, nous exposons une méthode d'optimisation de ces grammaires.

2 Les grammaires locales actuelles sous INTEX

2.1 Les graphes d'INTEX

Les grammaires locales utilisées sous INTEX ont la forme de graphes de transducteurs à états finis. Le système INTEX possède un éditeur de graphes FSGraph. Ces graphes ont un format nommé GRF. Nous donnons ci-dessous (figure 1) un exemple de graphe reconnaissant des dates qui nous servira tout au long de notre article. Il ne prétend en aucun cas décrire les dates de manière exhaustive. C'est un exemple symbolique qui illustrera nos propos. Les boîtes grisées représentent des appels à des sous-graphes (ici **En1999** et **EnDebutDeMatinee**). Les sous-graphes appelés dans cette grammaire se trouvent aux figures 2, 3 et 4. La transition vide est notée <E>. Le symbole # interdit l'espace. <le> désigne toutes les formes dérivées de la forme canonique *le*, soit *le*, *la*, *les*. L'expression entre guillemets *"2000"* reconnaît la séquence *2000* sans les guillemets et sans espace entre les chiffres. Par exemple, *2 000* ne sera pas reconnu car il y a un espace entre les chiffres *2* et *0*. Le graphe **DATE** reconnaît des expressions telles que *en 1867* ou *en début de matinée*. A noter que la transduction **( )ADV+date** permet cette insertion en sortie de l'application de ce graphe à un texte électronique. Il est possible d'utiliser des catégories grammaticales comme <N> qui désigne un nom, ou <V> qui désigne un verbe quelle que soit sa forme conjuguée.

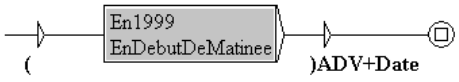

Figure 1 : **DATE** (GRF)

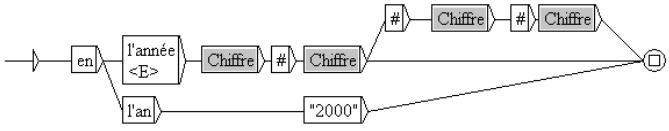

Figure 2 : **En1999** (GRF)

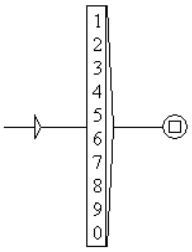

Figure 3 : **Chiffre** (GRF)

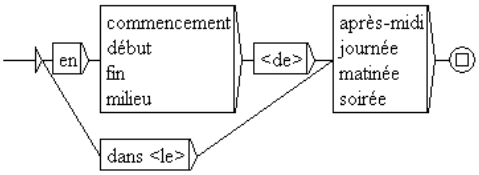

Figure 4 : **EnDebutDeMatinee** (GRF)

2.2 Le compilateur GRF2FST

Les graphes tels que nous les utilisons sont équivalents à des automates finis après remplacement des noms de sous-graphes par les sous-graphes correspondants. Sous INTEX, il existe un module qui transforme un graphe en transducteur à états finis (format FST). Ce module possède quelques opérations de nettoyage des graphes : suppression des transitions vides par exemple (voir 3.2). Il découpe les transitions en éléments simples. Par exemple, la transition étiquetée par *après-midi* est découpée en trois transitions étiquetées par *après*, - et *midi*. INTEX donne aussi la possibilité de déterminiser le transducteur. Notons que, pour de gros graphes, nous utilisons un module que nous avons développé en parallèle. Théoriquement, la représentation en transducteurs à états finis est ce qu'il y a de plus efficace en terme de vitesse d'application (E. Roche et al. 1997). Le FST déterminisé équivalent au graphe **DATE** (figure 1) est donné ci-dessous dans la figure 5 :

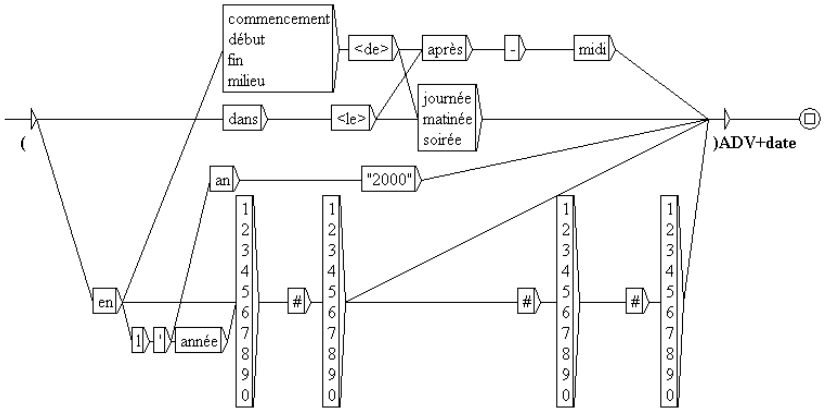

Figure 5 : **DATE** (FST)

2.3 Les limites du système actuel

Le système actuel pose certains problèmes qui, à terme, le rendront inadéquat. Nous dénombrons deux raisons principales pour changer de représentation de grammaires : l'une matérielle, l'autre théorique.

2.3.1 Limites matérielles

Alors que la place en mémoire prise par les graphes au format GRF ne posait pas de problème, elle en pose pour les FST. En effet, nous constatons une explosion exponentielle de la complexité en espace. Prenons le cas d'un graphe de M. Gross (1997) décrivant les verbes auxiliaires en anglais (**AUX**). Le format GRF prend 70 ko (pour une centaine de sous-graphes) et le format FST 33 Mo. Le FST déterminisé occupe même 50 Mo. Potentiellement, la complexité est exponentielle. A chaque remplacement d'un nom de graphe, on ajoute la place mémoire de ce graphe. Ce n'est pas le cas pour le GRF qui a une complexité additive. Illustrons notre propos par un exemple concret. Prenons un graphe décrivant une structure syntaxique du verbe *to give* : N_0 *give* N_1 *to* N_2 (figure 6). Supposons que les insertions (**Insert**) à l'intérieur de la phrase ne soient constituées que par des dates, soit le graphe **TIME** des expressions de temps (dates, durées) de M. Gross (plus de 50 Mo). De même, supposons que la place en mémoire occupée par les graphes reconnaissant les autres membres de la structure soit négligeable devant celles de **AUX** et **TIME**. Lors du passage en FST, **TIME** est dupliqué six fois (soit 300 Mo) et **AUX** deux fois (soit 100 Mo). Au total, le FST de ce graphe occupe 400 Mo en mémoire. Par ailleurs, cet exemple ne décrit qu'une seule structure du verbe *to give* qui n'est lui-même qu'un verbe parmi des milliers. Ainsi, il faudrait multiplier l'espace mémoire calculé ci-dessus par quelques centaines de milliers ; arrondissons ce coefficient à 100 000. Si un ordinateur personnel possédait 50 Go de mémoire virtuelle. Il faudrait au moins 800 ordinateurs pour stocker la totalité de la langue anglaise. Ce chiffre est aujourd'hui inacceptable. Nous montrons ainsi l'inadéquation de ce type de représentation.

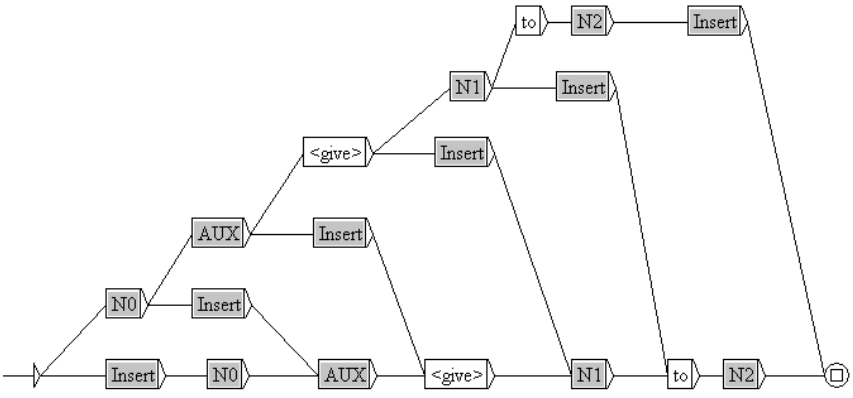

Figure 6 : **N0GiveN1ToN2**

2.3.2 Limites théoriques

Il est bien connu que la syntaxe du langage naturel ne peut pas toujours être décrite sous forme d'automates finis. Même s'ils sont très rares, il existe des cas de description "context-free" non "finite state" en linguistique, comme le célèbre et à peu près unique exemple :

> *La souris (que la petite fille (que Luc (que Marie aime +E) a prise en photo+E) regarde+ E) montre ses dents := P = N0 (Rel + E) V N1*

La description formelle de cet exemple possède une imbrication non bornée de la relative (*Rel*) et ainsi, la grammaire à utiliser n'est plus celle d'un automate fini, c'est une grammaire strictement context-free. Alors que théoriquement, le nombre de remplacements dans cette grammaire n'est pas borné, le compilateur d'INTEX (2.2) limite le nombre des remplacements à une constante. Si cette constante était égale à 2 par exemple, la phrase ci-dessus ne pourrait être reconnue. Le lecteur remarquera que, dans un cadre plus général, le formalisme "context-free" n'est plus suffisant pour décrire les langues naturelles : il faut utiliser des grammaires dépendantes du contexte.

3 Une évolution intermédiaire : le compilateur GRF2FST2

3.1 Le logiciel AGLAE

En prévision de l'explosion du nombre de grammaires et du temps nécessaire lors de leur application à des textes, nous avons développé en parallèle le logiciel AGLAE (S. Paumier à paraître) dont le but final est de se substituer à certains modules d'INTEX. AGLAE permet d'accélérer l'application des graphes sur de gros corpus. Il est, actuellement, jusqu'à 50 fois plus rapide. Ce logiciel contient deux modules bien distincts : un compilateur de graphes et un analyseur automatique. Le compilateur traite le graphe GRF fourni par l'utilisateur : par diverses opérations, il le transforme dans un nouveau format FST2 (3.2). L'analyseur applique le graphe compilé à un texte. La réunion des deux modules constitue le logiciel AGLAE. La compilation est invisible pour l'utilisateur. Comme dans cet article, nous ne nous intéressons qu'aux grammaires en elles-mêmes, nous nous limitons au module de compilation qui traite de représentation des graphes et d'opérations sur les grammaires locales.

3.2 Le format FST2 et le compilateur GRF2FST2

Le but du compilateur est d'éviter à l'analyseur de multiples opérations sur les graphes. Le compilateur prend en entrée un graphe GRF et délivre en sortie une nouvelle représentation de ce graphe, c'est le format FST2 que nous décrivons ci-dessous. Nous effectuons quelques traitements dont certains sont réalisés sur le FST : une normalisation des transitions puis un nettoyage et une déterminisation locale des graphes.

Dans un cadre théorique, hors linguistique, les graphes utilisés peuvent être vus comme des grammaires context-free décrites dans J.-M. Autebert et al. (1997). L'alphabet des éléments auxiliaires est l'ensemble des étiquettes indiquant les appels aux sous-graphes. Celui des éléments terminaux est l'ensemble des autres types d'étiquettes (catégories grammaticales, étiquettes lexicales, etc.). Ces deux alphabets sont clairement disjoints. Si *X* est un élément

auxiliaire, les règles correspondantes à *X* sont factorisées sous la forme d'un graphe nommé *X*. L'axiome de départ est le graphe principal. Jusqu'à présent, nous avons considérés nos grammaires comme de simples automates finis dans nos algorithmes. Vu les problèmes que cela risquait d'engendrer dans le futur (2.3), nous décidons de nous pencher sur une nouvelle représentation des graphes, équivalente à une grammaire context-free : le format FST2. Cette représentation a pour but de réduire sensiblement l'espace mémoire et de rester dans un environnement théorique correct. Contrairement au FST, nous gardons le principe d'appel à des sous-graphes comme dans le format GRF, utilisant ainsi le formalisme context-free.

Nous réalisons quelques traitements que l'on n'effectuait pas sur le GRF. Comme pour le FST, nous découpons les étiquettes en éléments terminaux. Chaque transition ne doit comporter qu'un seul élément terminal. On évite ainsi un découpage coûteux en temps lors de l'application des graphes. Nous ajoutons quelques améliorations par rapport au FST, notamment pour les expressions que l'on appelle entre guillemets (2.1). Prenons l'exemple du graphe 7. Il reconnaît l'unique expression *Jacques a dit*, et non *Jacques A DIT* qui comporte des majuscules. Dans le FST, l'expression *"Jacques a dit"* n'est pas découpée en éléments terminaux, elle est gardée telle quelle. Par contre, dans le FST2, un découpage est affectué (figure 8). Les espaces sont indiqués.

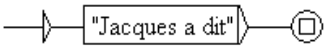

Figure 7 : **JacquesADit** (GRF et FST)

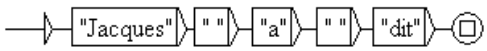

Figure 8 : **JacquesADit** (FST2)

Comme pour le FST, nous effectuons un nettoyage du graphe. Nous supprimons les transitions vides, les états non accessibles, les états non co-accessibles. Par ailleurs, nous déterminisons localement les grammaires. Les symboles auxiliaires sont alors considérés comme des transitions terminales : on ne rentre pas dans les sous-graphes. On reste à un niveau superficiel. La figure 9 présente le résultat de la compilation de **En1999** (figure 2) :

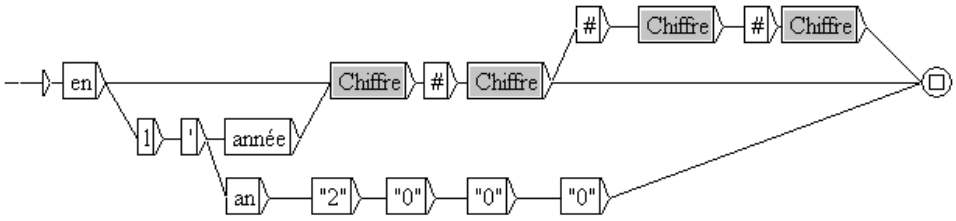

Figure 9 : **En1999** (FST2)

La représentation est intermédiaire entre FST et GRF. En effet, comme pour le GRF, nous avons des appels à des sous-graphes, ici **Chiffre**. Comme pour le FST, les transitions à étiquettes multiples sont décomposées : *l'année*, *l'an* et, plus particulièrement pour le FST2, *"2000"*. Le FST2 est déterminisé localement : nous factorisons les transitions *l'*. L'espace mémoire occupé par le FST2 est du même ordre que le GRF. Le graphe **TIME** au format FST2 prend environ 100 ko (plus de 50 Mo pour le FST). La différence est claire. Nous allons voir, dans la partie suivante, quelques moyens d'optimiser cette représentation.

4 Optimisation des graphes : la déterminisation Look-Ahead

4.1 Les limites du FST2

Le résultat obtenu par le compilateur GRF2FST2 peut être amélioré. Nous avons appliqué, à l'aide d'AGLAE, notre graphe **AUX** des verbes auxiliaires (2.3.1) sur un corpus constitué d'une année du journal électronique Le Monde (110 Mo). Cette application a pris une heure. Puis, nous avons adapté AGLAE afin qu'il accepte en entrée le FST de ce graphe. L'application prend, dans ce cas, trois-quart d'heure. La différence d'efficacité entre FST2 et FST est nette. Nous expliquons ce résultat par deux phénomènes. Premièrement, la déterminisation réalisée par le compilateur reste locale. Deuxièmement, l'utilisation de sous-graphes empêche un accès direct aux éléments terminaux. Dans les parties suivantes, nous optimisons le FST2 compilé grâce à deux méthodes : une déterminisation locale plus approfondie (4.2) et une remontée des transitions terminales (4.3). L'inconvénient de ces deux méthodes est d'accroître l'espace mémoire nécessaire pour stocker les graphes. Par des expérimentations, il faudra trouver un compromis entre vitesse d'accès et place mémoire.

4.2 Une déterminisation locale approfondie

Dans la section précédente, nous avons mentionné que le compilateur réalisait une déterminisation locale des graphes. Si nous regardons les graphes **En1999** (figure 2) et **EnDebutDeMatinee** (figure 4), nous constatons qu'ils commencent tous deux par la transition étiquetée par *en*. Alors que la déterminisation de **DATE** au format FST (figure 5) met cette transition en facteur pour les deux graphes, la déterminisation locale de **DATE** ne peut plus le faire. Ce phénomène est problématique pour les gros graphes. Prenons l'exemple du graphe **TIME** de M. Gross, qui comprend une centaine de sous-graphes, la plupart mis en parallèle les uns par rapport aux autres. En faisant remonter les transitions terminales partant de l'état initial, au niveau du graphe principal, nous dénombrons 29 618 transitions alors que le nombre total d'étiquettes est 1 134. **TIME** est donc largement non déterministe. En effet, s'il était déterministe, le nombre de transitions terminales initiales serait au plus égal au nombre d'étiquettes. Nous expliquons ainsi une grosse partie de la perte de temps constatée en 4.1. Une solution est d'améliorer la déterminisation. Au lieu de rester en superficie, nous entrons partiellement dans les sous-graphes et nous regardons leurs premières transitions. Il est clair que nous ne pouvons pas déterminiser totalement : risque d'explosion de la mémoire et, dans le cas de graphes récursifs, de boucle infinie. Nous réalisons une déterminisation partielle. Nous fixons un paramètre *n*, désignant la pénétration horizontale maximale au-delà de laquelle la déterminisation s'arrête. Prenons l'exemple, du graphe **President** de la figure 10. Si l'on regarde les premières transitions des deux sous-graphes **PresidentEco** (figure 10) et **PresidentPol** (figure 11), on voit que l'on peut factoriser une bonne partie des deux sous-graphes. Nous donnons le résultat **PresidentDet2** de la déterminisation locale approfondie en prenant $n = 2$ (figure 13). Nous ne donnons pas les sous-graphes créés lors de la déterminisation partielle car ils peuvent être facilement retrouvés par le lecteur à l'aide des graphes de départ (figures 10, 11 et 12). On s'aperçoit que de nombreux sous-graphes sont créés : la taille mémoire augmente. Il faut donc éviter de prendre un *n* trop grand.

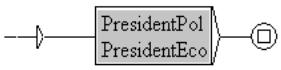

Figure 10 : **President**

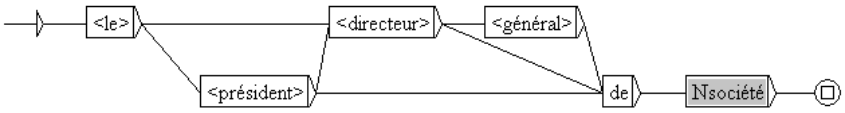

Figure 11 : **PresidentEco**

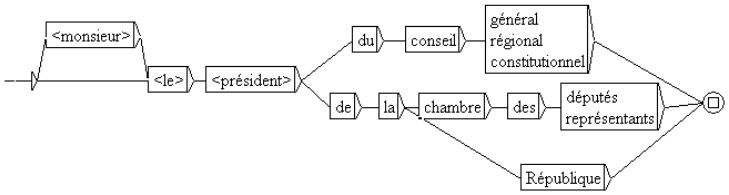

Figure 12 : **PresidentPol**

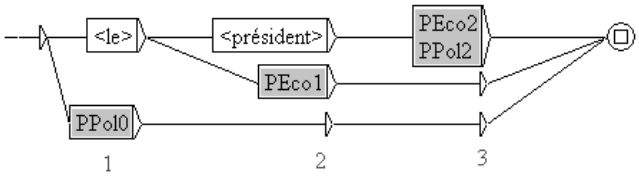

Figure 13 : **PresidentDet2**

4.3 Remontée des éléments terminaux

Lors de l'application des grammaires aux textes, la coïncidence ne se fait qu'entre les mots du texte et les éléments terminaux des graphes. S'il y a plusieurs couches d'appels à des sous-graphes, il faut aller chercher les éléments terminaux pour pouvoir faire une comparaison. Il faut éviter la présence des éléments auxiliaires, c'est-à-dire des appels à des sous-graphes. Ainsi, comme les premières transitions d'un graphe jouent un rôle de filtre, nous réalisons une variante de la mise en forme normale de Greibach (S. Greibach 1965) appliquée à des graphes d'automates finis. Cette méthode consiste à faire remonter les premières transitions terminales au niveau du graphe principal. Ainsi, le graphe **President** (figure 10) est transformé en le graphe **PresidentGr** ci-dessous (figure 14). Nous constatons que ce graphe n'est pas déterministe. Comme précédemment, les sous-graphes peuvent être retrouvés par le lecteur.

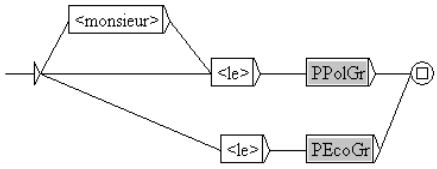

Figure 14 : **PresidentGr**

4.4 La déterminisation Look-Ahead

A partir de la discussion précédente, nous décidons de développer un nouveau module appelé Déterminisation Look-Ahead (DLA). La méthode utilisée consiste à fusionner les deux approches données en 4.2 et 4.3. Soit *n* un paramètre que nous fixons. Nous terminalisons et déterminisons partiellement le graphe en entrée, jusqu'à la pénétration horizontale maximale *n*. Désormais, *LA*-déterminiser sera effectuer une DLA. Pour *n=2*, la DLA donnerait le résultat suivant (figure 15) sur le graphe **President** (figure 10) :

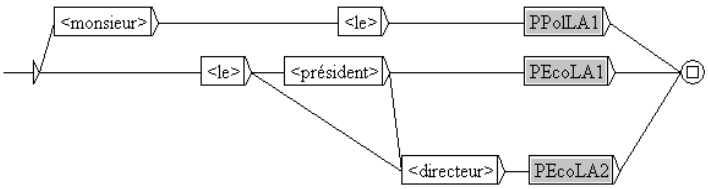

Figure 15 : **President_LA2**

Nous avons *LA*-déterminisé le graphe **TIME** en faisant varier le paramètre *n*. Ensuite, nous avons appliqué les graphes résultats à un corpus littéraire de 1 Mo constitué du texte *Portrait of a Lady* de Henry James, avec le logiciel AGLAE (4.3). Quelques résultats de l'expérience sont donnés dans le tableau ci-dessous (figure 16). Notons que la DLA au paramètre *n=0* constitue le graphe d'entrée. Nous constatons que les résultats dépendent de la nature des graphes. La mémoire peut rapidement exploser comme pour **TIME** (dès *n = 2*). Les performances en temps sont nettement accélérées avec *n* croissant.

FST2	TIME			AUX			
N	0	1	2	0	1	2	3
Espace mémoire	127 ko	342 ko	11 Mo	60 ko	117 ko	142 ko	500 ko
Temps d'application au corpus	420 s	50 s	9 s	9 s	4 s	3 s	2 s

Figure 16 : résultats de l'expérience

Nous pensons qu'il faut ajouter des paramètres liés à la nature du graphe, afin de préciser l'optimisation. Nous voulons *LA*-déterminiser tous les sous-graphes sous certaines conditions et pas uniquement le graphe principal. Par exemple, nous souhaitons mettre une condition sur le nombre de sous-graphes appelés au début du graphe que l'on *LA*-déterminise. Si cette condition est vérifiée, on *LA*-déterminise normalement jusqu'à *n =1* ou *2* ; sinon, on ne fait rien de spécial. Cette procédure est en cours d'élaboration. Nous pensons encore améliorer sensiblement les performances.

5 Conclusion et perspectives

La quantité de grammaires locales nécessaires aux analyses remet en cause les performances des outils actuels. Nous avons conçu un système utilisant une nouvelle représentation équivalente à une grammaire context-free et de nouveaux traitements. Nous avons décrit une méthode d'optimisation paramétrée des graphes : la Déterminisation Look-Ahead (DLA). A terme, nous envisageons d'optimiser automatiquement les graphes. Par le développement d'un analyseur automatique de graphes, nous pourrions pré-calculer les paramètres de la DLA. Ces représentations et optimisations des graphes nous paraissent indispensables dans le cadre du LADL. Par ailleurs, nous pensons que ces travaux pourraient être utiles à d'autres applications linguistiques utilisant des grammaires de grande taille.

Remerciements

Nous tenons à remercier M. Gross pour ses conseils et S. Paumier pour sa collaboration active.

Références

Autebert J., Berstel J., Boasson L. (1997), Context-free languages and Pushdown Automata, in G. Rozenberg and A. Salomaa Eds., *Handbook of Formal Languages, Vol.1: Word, Language, Grammar*, Springer-Verlag, Berlin, pp. 125-213

Greibach S. (1965), A new normal form theorem for context-free phrase structure grammars, *Journal of ACM*, Vol.12, pp. 42-52

Gross M. (1997), The Construction of Local Grammars, in E. Roche and Y. Schabes Eds., *Finite State Language Processing*, Cambridge, Mass., The MIT Press, pp. 329-352.

Leclère C., Subirats-Rüggeberg C. (1991), A bibliography of studies on lexicon-grammar, *Lingvisticæ Investigationes*, Vol. XV:2, pp.347-409.

Paumier S., (à paraître), Nouvelles méthodes pour la recherche d'expressions dans de grands corpus, *RISSH*.

Revuz, D. (1991), *Dictionnaires et lexiques : méthode et algorithmes*, Thèse de doctorat, Paris, Université Paris 7.

Roche E. (1993), *Analyse syntaxique transformationnelle du français par transducteurs et lexique-grammaire*, Thèse de doctorat, Paris, Université Paris 7.

Roche E., Schabes Y. (1997), Introduction, in E. Roche and Y. Schabes, *Finite State Language Processing*, Cambridge, Mass., The MIT Press, pp. 1-66.

Silberztein M. (1993), *Dictionnaires électroniques et analyse automatique de textes : Le système INTEX*, Paris, Masson.