Nouvelles méthodes pour la recherche d'expressions dans de grands corpus

Sébastien PAUMIER

Abstract : As graphs and corpora become larger every day, some specific tools become necessary. In this paper, we present an additional tool to INTEX for processing large corpora and graphs. Based on new methods specifically developed to answer this problem, it is faster than the existing programme in many situations and makes easier the recognition of expressions. We describe here the main features of our algorithms.

Keywords: graph, algorithm, recognition of expressions, large corpora.

Mots clés : graphe, algorithme, reconnaissance d'expressions, grands corpus.

1. Introduction

Le traitement automatique des langues naturelles fait l'objet de méthodes très différentes. Le plus souvent, il est basé sur des procédures formelles du type GREP, LEX ou AWK. Mais il se développe des approches ayant recours aux propriétés linguistiques des langues traitées. Celles-ci sont consignées dans des dictionnaires et grammaires électroniques. Les grammaires construites au LADL reposent sur un principe de description des langues au moyen d'automates finis. Ce formalisme extrêmement souple permet de représenter très simplement la plupart des phénomènes linguistiques. Pour être efficace, cette approche doit s'appuyer sur des descriptions les plus exhaustives possible, ce qui nécessite l'étude de corpus volumineux.

✉ Sébastien PAUMIER, Laboratoire d'Automatique Documentaire et Linguistique,
Université de Marne la Vallée
Bureau 4B098, bâtiment Copernic, 5 bd Descartes,
77457 Champs sur Marne, Marne la Vallée
Tel : (+33) (0)1 44 27 56 95 e-mail : paumier@univ-mlv.fr

On doit donc être en mesure de manipuler de gros textes dans un temps raisonnable. Une fois constituées, les bibliothèques de graphes doivent pouvoir être appliquées rapidement, ce qui n'est pas trivial étant données leurs tailles. Il faut donc des méthodes permettant de manipuler simultanément de gros graphes et de gros corpus. Nous présenterons les méthodes que nous avons utilisées et qui ont conduit à la réalisation du logiciel AGLAE. Ce programme étant destiné aux utilisateurs du logiciel INTEX [1], quand nous parlerons de graphes, de dictionnaires ou de textes, il s'agira des ressources gérées par INTEX. De même nous reprendrons les conventions de ce système.

2. Les corpus

2.1. Taille des corpus

L'expansion d'Internet aidant, il est aujourd'hui facile de se procurer de gros corpus [2]. Afin de pouvoir traiter de telles masses de texte, notre premier souci a été de découper ces textes en blocs. La manipulation des corpus bloc par bloc les rend accessibles à des machines d'une puissance de calcul réduite. En outre, un tel découpage permettra, à terme, de traiter un texte par des procédures parallèles, chaque processeur traitant indépendamment un ou plusieurs blocs.

2.2. Représentation des corpus

Contrairement à d'autres systèmes, nous n'avons pas choisi la représentation du texte par un index. Cette méthode est courante dans les applications commerciales car elle présente l'avantage d'accélérer la recherche par mot clé. La représentation des graphes que nous utilisons ne nous permettrait de bénéficier d'un index du texte que dans des cas triviaux (recherche d'un ou deux mots), cas que gèrent sans problème les systèmes existants. Nous avons donc préféré conserver une structure de texte linéaire. Partant du principe que notre programme recourt à des données linguistiques, nous avons pris comme unité non pas le caractère mais l'unité lexicale (token). Ainsi chaque

[1] La fonctionnalité qui nous intéresse dans cet article est l'application d'un graphe décrivant une grammaire locale à un texte.

[2] Par gros corpus, nous entendons des documents de l'ordre du giga-octet.

token se voit attribuer un entier et le texte est représenté par une suite d'entiers.

Considérons le texte = {S}*Un chat est un chat.*

Unité lexicale	{S}	Un	*espace*	chat	est	un	.
Numéro	0	1	2	3	4	5	6

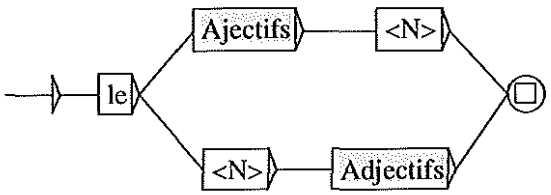
Le fichier représentant le texte contiendra donc la suite :

0 1 2 3 2 4 2 5 2 3 6

On notera qu'il est tenu compte des différences minuscules/ majuscules : *Un* et *un* correspondent chacun à un entier différent. Comme on le verra, cette représentation permet d'optimiser un certain nombre d'opérations ultérieures.

3. Compilation des graphes

La représentation FST (finite-state transducer) repose sur une transformation des graphes où chaque appel à un sous-graphe est remplacé par le sous-graphe lui-même, comme on peut le voir sur l'exemple suivant :

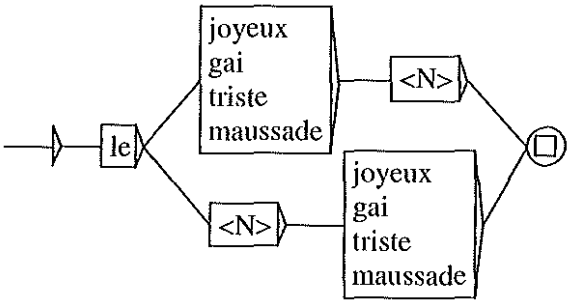


Graphe 1

Graphe 2 : Adjectifs.grf

La compilation du graphe 1 donne en FST l'équivalent du graphe 3, ce qui introduit des problèmes de redondances qui peuvent faire exploser la taille des fichiers FST. Afin d'éviter ce problème, nous

avons conservé la structure en sous-graphes. L'avantage est la taille réduite de la représentation, l'inconvénient est qu'elle ne peut pas être complètement déterminisée, ce qui peut multiplier par 3 ou 4 le temps des calculs. Nous nous sommes pour l'instant contenté de déterminiser localement chaque sous-graphe, mais il sera sans doute possible d'améliorer les choses en développant une déterminisation explorant les premières branches des sous-graphes.



Graphe 3

La compilation a ceci d'intéressant qu'elle transforme chaque sous-graphe en un automate dont les transitions sont étiquetées par des unités lexicales. On pourra donc remplacer ces unités lexicales par leurs entiers associés, ainsi qu'on l'a fait pour le texte. Le fait de manipuler des entiers au lieu de chaînes de caractères va déjà nous permettre de réduire le nombre d'opérations d'un facteur f où f est le nombre moyen de lettres par mot (5,7) calculé sur l'année 1994 du journal *Le Monde*, c'est-à-dire sur 120 Mo de texte.

4. Traitement des étiquettes

4.1. Prétraitements

Une fois le graphe compilé et chargé, on va effectuer deux prétraitements sur les étiquettes afin d'accélérer l'exploration du graphe lors de son application au texte.

Le premier de ces traitements consiste à remplacer chaque transition étiquetée d'un mot ou d'un lemme (ou entrée de dictionnaire) par la liste de toutes ses variantes. Dans le cas d'un lemme comme <manger.V>, on extrait du dictionnaire du texte toutes les formes fléchies

(conjuguées) dont la forme canonique est ce lemme. On ajoute ensuite à cette liste toutes les variantes minuscules/majuscules.

Ainsi, l'étiquette <manger.V> sera remplacée par la liste suivante : '*mange*', '*Mangerons*', '*MANGER*', '*mangeait*', etc.

Chaque forme est ensuite remplacée par son numéro.

Le second prétraitement porte sur les formes entre angles ne contenant ni formes fléchies, ni lemmes. Les symboles méta comme <MAJ>, <MIN> [1], etc. sont remplacés par des codes spéciaux. On a pré-calculé pour chaque mot du texte quels métas il vérifiait. Lors de l'exploration du graphe, la comparaison entre un méta et un mot du texte se fait très simplement : si le nœud courant du graphe est un méta, on regarde si mot du texte vérifie ce méta au moyen d'une comparaison d'entiers.

Les formes du type <N:ms>, que nous appellerons patterns, sont extraites du graphe et numérotées. Lors du chargement des dictionnaires, on vérifie pour chaque mot quelles formes il vérifie.

Par exemple : on a dans le graphe les patterns suivants : <V>, <N+z1> et <N:fs:ms> qui désignent respectivement les verbes, les noms du langage courant (couche z1) et les noms singuliers au masculin ou au féminin.

Dans le dictionnaire du texte on trouve les lignes suivantes :

garde,garde.N+z1:fs
garde,garde.N+z1:ms:fs
garde,garder.V+z1:P1s:P3s:S1s:S3s:Y2s

Si on numérote ainsi les catégories :

<V>=0, <N+z1>=1 et <N:fs:ms>=2

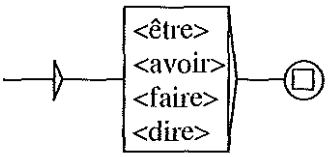
alors le code de *garde* est l'ensemble {0,1,2}

Un mécanisme efficace de gestion d'ensembles nous permet alors de dire si le mot courant du texte vérifie un pattern grâce à une comparaison d'entiers.

[1] Les codes <MAJ> et <MIN> désignent respectivement les mots tout en majuscules et les mots tout en minuscules.

4.2. Compactage du graphe

À ce stade, le graphe a été transformé de façon à ce que son exploration ne nécessite que des comparaisons d'entiers. Une procédure de compactage va nous permettre d'accélérer considérablement les calculs pour un coût relativement réduit. Considérons le graphe suivant :



Graphe 4

Une fois prétraité, comme décrit en 4.1, ce graphe pourra comporter plus d'une centaine de formes (chacun de ces quatre verbes possède 39 formes conjuguées qui peuvent toutes avoir des variantes minuscules/majuscules). Or, dans l'automate équivalent à ce graphe, cela correspond à plus de 100 transitions partant d'un état pour arriver à un second état. Cela nous a conduit très naturellement à regrouper ces formes dans une classe.

Si on attribue à cette classe le numéro *i*, il suffit de noter que chaque mot de la liste appartient à la classe *i* et de remplacer la liste des 100 transitions par une seule indiquant qu'il s'agit de la classe *i*. Lors de l'exploration du graphe compacté, il suffit d'appliquer une fonction *Appartient(mot,classe)* pour le parcourir de façon équivalente au graphe d'origine. Grâce à une gestion d'ensembles similaire à celle utilisée pour gérer les patterns, nous avons pu réduire cette fonction à une comparaison d'entiers. Cette méthode étant rentable dès 2 transitions, le gain est d'autant plus grand que les graphes contiennent d'importantes listes de mots, comme par exemple les graphes dictionnaires.

Cette technique ne coûte qu'un bit pour exprimer qu'un mot appartient à une classe ; on peut donc l'appliquer sans problème sur de gros ensembles de données.

5. Conclusion

Compte tenu de la taille des données que les outils seront amenés à traiter, il nous a paru primordial de s'attacher à une optimisation maximum des programmes. Le programme AGLAE disponible sur le site du LADL (http://ladl.univ-mlv.fr/tools/aglae/aglae.html) résulte de la mise en œuvre de toutes les techniques décrites dans cet article. Testé sur un corpus de 120 Mo à l'aide de séries de plusieurs centaines de graphes, il a mis en évidence l'efficacité des méthodes proposées par comparaison avec le système INTEX, pouvant aller jusqu'à 600 fois plus vite dans des cas favorables. Les graphes d'expressions numériques de Matthieu Constant [1] ont pu ainsi être appliqués à une année du journal *Le Monde* en une heure.

Comme ce programme a été conçu pour traiter de gros jeux de données sans index, il est moins performant que les systèmes existants quand les expressions recherchées sont rares et situées en fin de texte. Si l'on considère un texte de 100 Mo où la seule occurrence d'un mot apparaît dans la dernière phrase, rechercher ce mot à l'aide d'un parcours linéaire comme celui que nous utilisons est bien moins efficace qu'une recherche utilisant un index du texte.

Enfin, l'étude d'une déterminisation plus efficace nous permettra sans doute d'améliorer encore les performances avant de devoir passer la main à la puissance mécanique du calcul parallèle.

Bibliographie

PAUMIER (Sébastien) : 2000, *Recherche d'expressions dans de grands corpus : le système AGLAE*, Mémoire de DEA (Université de Marne-la-Vallée).

PAUMIER (Sébastien), CONSTANT (Matthieu), *A corpora processing software : AGLAE.* [http://ladl.univ-mlv.fr/tools/aglae/aglae.html]

ROCHE (Emmanuel), SCHABES (Yves) : 1997, *Finite-state language processing* (The MIT Press).

SILBERZTEIN (Max) : 1993, *Dictionnaires électroniques et analyse automatique des textes. Le système INTEX* (Paris : Masson).

[1] L'automate FST équivalent à ces graphes contient 100 000 états et 700 000 transitions.