Inférence grammaticale guidée par clustering

Noémie-Fleur Sandillon-Rezer
CNRS, Esplanade des Arts et Métiers, 33402 Talence
LaBRI, 351 Cours de la Libération, 33405 Talence
nfsr@labri.fr

RÉSUMÉ

Dans cet article, nous nous focalisons sur la manière d'utiliser du clustering hiérarchique pour apprendre une grammaire AB à partir d'arbres de dérivation partiels. Nous décrirons brièvement les grammaires AB ainsi que les arbres de dérivation dont nous nous servons comme entrée pour l'algorithme, puis la manière dont nous extrayons les informations des corpus arborés pour l'étape de clustering. L'algorithme d'unification, dont le pivot est le cluster, sera décrit et les résultats analysés en détails.

ABSTRACT

Clustering for categorial grammar induction

In this article, we describe the way we use hierarchical clustering to learn an AB grammar from partial derivation trees. We describe AB grammars and the derivation trees we use as input for the clustering, then the way we extract information from Treebanks for the clustering. The unification algorithm, based on the information extracted from our cluster, will be explained and the results discussed.

MOTS-CLÉS : grammaires catégorielles, clustering hiérarchique, inférence grammaticale.

KEYWORDS: categorial grammars, hierarchical clustering, grammatical inference.

1 Introduction

Le but de cet article est de présenter une nouvelle méthode d'inférence grammaticale. En effet, nous utilisons en entrée de notre algorithme des arbres de dérivations d'une grammaire AB partiellement remplis, et l'algorithme est ensuite guidé par le clustering pour savoir quelles variables vont être unifiées. L'idée de base est que les mots qui sont dans des contextes similaires doivent avoir des types similaires.

L'inférence grammaticale appliquée aux grammaires catégorielles peut être classée en trois catégories distinctes, en fonction des méthodes employées et des structures d'entrée.

La méthode décrite par Adriaans (Trautwein *et al.*, 2000; Adriaans, 1999) a pour point de départ des phrases sans structure. Bien qu'elle fonctionne dans la plupart des cas, ce genre de méthode d'inférence rencontre des problèmes avec des phrases permettant plusieurs analyses syntaxiques qu'on ne peut pas distinguer et qui sont fondées seulement sur la chaîne des mots

-par exemple l'attachement des syntagmes prépositionnels. Il semble donc logique, étant donné que cette information est généralement annotée dans des corpus arborés, de l'utiliser lors de l'apprentissage.

Des structures partielles sont utilisées dans les méthodes de Buszkowski et Penn (Buszkowski et Penn, 1990) ou Kanazawa (Kanazawa, 1998). Ces méthodes sont clairement dans la lignée du paradigme de Gold (Gold, 1967). Les structures d'entrée sont décrites ultérieurement, section 3. La sortie de ces algorithmes donne soit une grammaire rigide [1] (algorithme de Buszkowski et Penn) soit une grammaire k-valuée [2] (algorithme de Kanazawa). Une grammaire rigide n'est pas représentative d'un langage naturel, et dès $k \geq 2$, l'unification devient un problème NP-dur (Costa-Florêncio, 2001). Outre ce fait, comme k n'est pas connu par avance, il convient de trouver la valeur optimale, ce qui est particulièrement complexe. On doit alors appliquer une recherche dichotomique pour trouver k, partant du principe que celui-ci se situe au départ entre 1 et ∞. L'idée est de prendre une valeur arbitraire qui semble raisonnable, de tester. Si cela fonctionne, on divise la valeur par deux et on tente à nouveau, sinon on la multiplie par deux en partant du principe que la bonne valeur devra être supérieure ou égale à $k + 1$. Il est à noter, cependant, que même une grammaire 2-valuée ne peut pas représenter une langue naturelle : l'expérience avec les grammaires extraites montre que le nombre maximum de types par mot est grand, et de nombreux mots fréquents (déterminants, conjonctions de coordinations) possèdent plus de quarante types (Hockenmaier et Steedman, 2007; Sandillon-Rezer et Moot, 2011).

Enfin, les méthodes utilisent des structures totalement définies, comme celle d'Hockenmaier (Hockenmaier, 2003), qui construit une grammaire catégorielle combinatoire, ou notre méthode (Sandillon-Rezer et Moot, 2011), qui utilise un transducteur d'arbres généralisé pour transformer des arbres syntaxiques en arbres de dérivation d'une grammaire AB (Lambek, 1958). C'est la sortie de notre transducteur qui nous servira de standard d'évaluation ainsi que d'entrée après modification des arbres pour notre algorithme d'inférence grammaticale.

Dans cet article nous combinons les méthodes du second type avec des structures partielles (agrémentées de certaines informations provenant des corpus) et du clustering. Nous évaluons à la fois la complexité du problème et la qualité du lexique obtenu. Le clustering est effectué en utilisant une mesure de similarité fondée sur le contexte local du mot, qui est directement extrait des arbres syntaxiques.

Les arbres de dérivation sont extraits de corpus annotés. Nous utilisons deux corpus différents en guise de base : le corpus de Paris VII (Abeillé *et al.*, 2003) et Sequoia (Candito et Seddah, 2012). Les deux corpus sont annotés de manière syntaxique par les mêmes protocoles d'annotation (Abeillé et Clément, 2003). Les principales différences entre les deux corpus résident dans le nombre de phrases et l'origine de celles-ci. Le corpus de Paris VII est composé de 12351 phrases qui proviennent d'une sélection d'articles du journal *Le Monde*, et Sequoia est composé de 3204 phrases venant de différents horizons, comme Wikipedia, le journal *L'Est Républicain* ou encore des notices médicales. La figure 1 donne un exemple d'arbre syntaxique. Les noeuds pré-terminaux contiennent les POS-tag [3] du mot, les autres noeuds internes contiennent le type syntagmatique du sous-arbre et les feuilles représentent les mots de la phrase.

Etant donné que le format des annotations ne correspond pas aux arbres de dérivation d'une grammaire AB, nous utilisons le transducteur d'arbres généralisé pour transformer les arbres du

1. Une grammaire catégorielle rigide force les mots du lexique à n'avoir qu'un seul type.
2. Une grammaire k-valuée permet aux mots d'un lexique d'avoir k types au maximum.
3. les annotations *parties du discours*

corpus de Paris VII et de Sequoia en arbres de dérivation.

Nous commencerons par décrire la grammaire AB générée par le transducteur généralisé, ensuite nous rappellerons le principe général de l'algorithme d'unification pour les grammaires AB et décrirons celui que nous utilisons. Dans la section quatre, nous décrirons le format de vecteurs utilisés pour l'étape de clustering. L'évaluation de notre méthode suivra, ainsi qu'une discussion sur les extensions possible de ce travail.

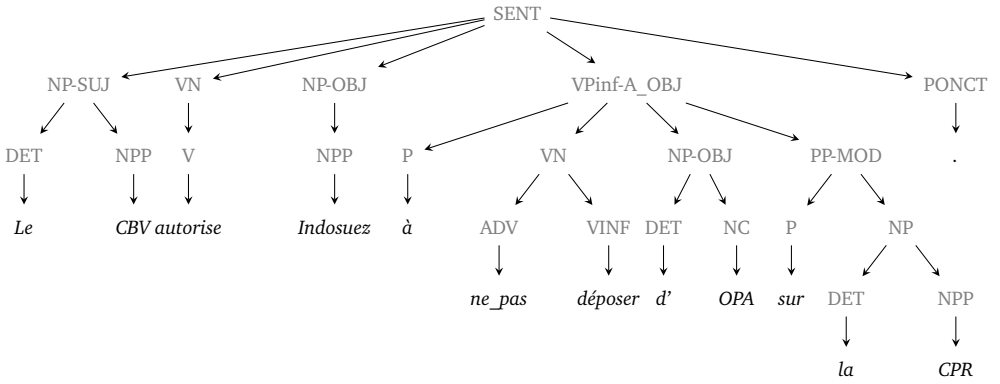

FIGURE 1 – Exemple d'arbre syntaxique du corpus de Paris VII.

2 Arbres de dérivation d'une grammaire AB

Les grammaires AB ont été définies séparément par Ajdukiewicz (Ajdukiewicz, 1935) et Bar-Hillel (Bar-Hillel, 1964) à partir du coeur des grammaires catégorielles et sont à présent considérées comme une sous-partie du calcul de Lambek (Lambek, 1958) et des grammaires catégorielles combinatoires [4] (Hockenmaier et Steedman, 2007). Les grammaires AB ont seulement deux règles d'éliminations, comme montré tableau 1.

$$\frac{A/B \quad B}{A}\ [/E] \qquad \frac{B \quad B\backslash A}{A}\ [\backslash E]$$

TABLE 1 – The elimination rules for AB grammar

Les arbres de dérivation d'une grammaire AB représentent l'application successive des règles d'éliminations.

Notre transducteur généralisé, qui correspond à une version modifiée d'un transducteur descendant, transforme les arbres syntaxiques des deux corpus en dérivations d'une grammaire AB. La figure 2 montre un exemple de sortie du transducteur correspondant à l'arbre syntaxique de la

4. Il est à noter cependant que nous suivons la convention déterminée par Lambek d'avoir toujours la catégorie qui sert d'argument sous le slash.

figure 1. Moins de 1.650 règles de transduction sont nécessaires pour convertir 92% du corpus de Paris VII (93% de Sequoia).

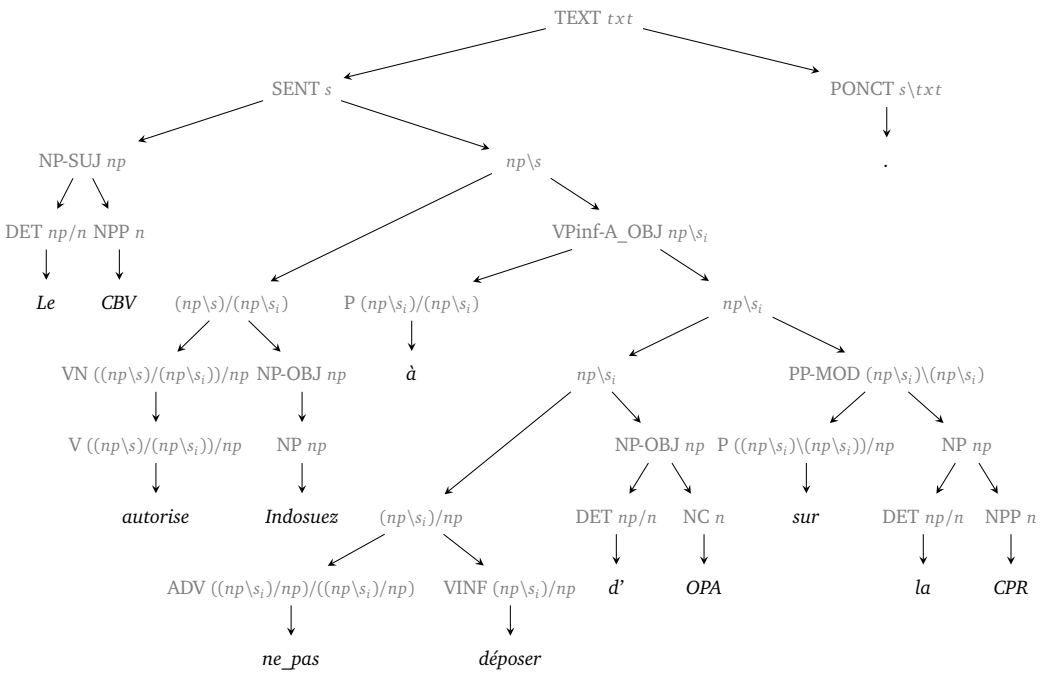

FIGURE 2 – Sortie du transducteur. Les informations provenant de l'arbre syntaxique sont toujours présentes. Il est à noter que sur la sortie réelle, les types sont aussi présents dans les feuilles ; ils sont hérités des noeuds pré-terminaux.

Le transducteur utilise quelques types atomiques pour l'étape de transduction : *np, n, s, txt, pp, cs,* cl_r. Cela correspond respectivement, à : un syntagme nominal, un nom commun, une phrase, un "texte" (une phrase avec une ponctuation finale), un syntagme prépositionnel, une clause subordonnée et un clitique réflexif. En plus, nous utilisons les types $np\backslash s_p$ et $np\backslash s_i$ pour les syntagmes participiaux et infinitivaux. Cependant, le transducteur est relativement modulaire. Chacun peut créer un ensemble de règles pour binariser les arbres et le transducteur vérifiera que les arbres sont bien binaires à la sortie et que les types sont cohérents au sens de Ajdukiewicz, c'est à dire qu'on a bien à chaque étape une application d'une des règles d'élimination.

Nous utiliserons ces types pour initialiser nos arbres avant l'étape d'unification ; la description du format d'entrée est effectuée dans la section 3.

3 Inférence grammaticale

Un algorithme d'inférence grammaticale bien connu est celui décrit par Buszkowski et Penn (Buszkowski et Penn, 1990). Pour apprendre une grammaire rigide, cela ne pose pas de problème

(voir algorithme 1) : soit les types du lexique peuvent être unifiés jusqu'à ce qu'il n'y en ait plus qu'un par mot, soit l'algorithme échoue. Pour apprendre une grammaire k-valuée (Kanazawa, 1998) il y a besoin du même format d'entrée, comme montre la figure 3. Le coeur du problème avec les grammaires k-valuées est de décider quels types doivent être unifiés, car la meilleure unification possible ne peut être décidée que d'un point de vue global. C'est pour cela que ce problème d'inférence grammaticale est NP-dur (Costa-Florêncio, 2001) dès que $k \geq 2$. Il est également important de noter que k n'est généralement pas connu d'avance, ce qui complique la résolution de l'algorithme.

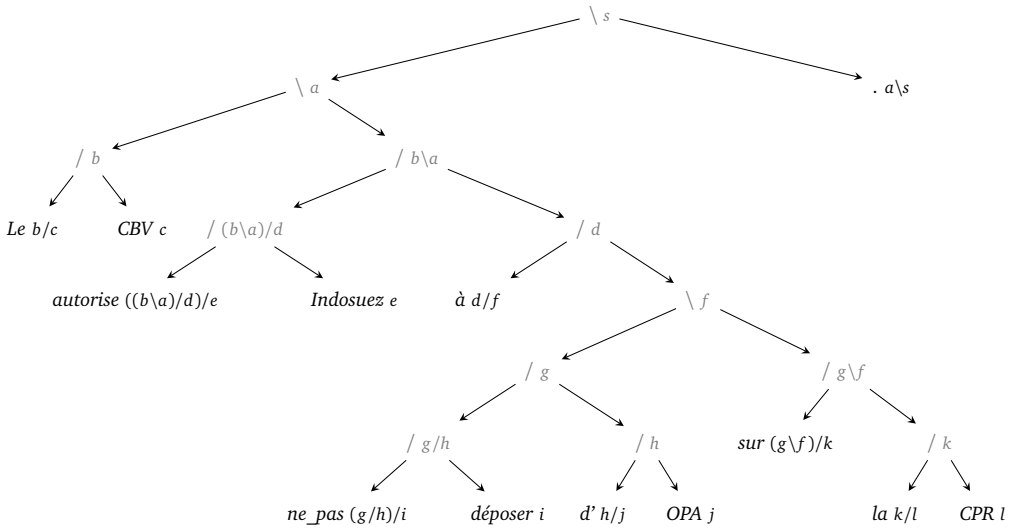

FIGURE 3 – Exemple d'entrée pour les deux algorithmes d'unification.

Données: arbres dont la racine est étiquetée s, les noeuds internes / ou \, et les feuilles avec des variables
Résultat: une grammaire rigide
création d'un lexique de mots contenant toutes les variables qui leurs sont liées ;
tant que *chaque mot a plusieurs types qui lui sont liés* **faire**
  rechercher l'unificateur le plus généraliste, de manière à réduire globalement le nombre de variables liées aux mots;
  si *l'unification n'est pas possible* **alors** l'algorithme échoue;
fin

Algorithm 1: Algorithme d'apprentissage d'une grammaire rigide.

Pour illustrer le problème de l'unification, il suffit de prendre deux phrases du corpus de Paris VII : *Nous avons de plus en plus de monde dans les DOM-TOM* et *Le gouvernement n'avait ni écrit ni choisi cet accord dont nous avons hérité*. Le lexique avant unification est décrit tableau 2. En appliquant l'algorithme de Buszkowski et Penn, le verbe *avons* aura deux types qui se ressemblent : $(d\backslash c)/e$ et $(w\backslash v)/x$. En effet, à chaque fois *avons* prend deux arguments, l'un à sa gauche et l'autre à sa droite. Cependant, nous ne pouvons pas unifier ces deux types, parce que dans le premier cas

l'argument de droite est un groupe nominal et dans le second cas un participe passé ; en outre nous ne souhaitons pas que les deux aient le même type, pour éviter des phrases agrammaticales. Pour ces deux phrases, nous avons donc besoin au minimum d'une grammaire 2-valuée, et *avons* doit avoir deux types différents : $(np\backslash s)/np$ et $(np\backslash s)/(np\backslash s_p)$.

nous	d, w	avons	$(d\backslash c)/e$, $(w\backslash v)/x$	de_plus_en_plus	e/f
de	f/g	monde	g	dans	$(c\backslash a)/h$
les	h/i	DOM-TOM	i	le	l/m
gouvernement	m	n'	$((l\backslash k)/n)/p$	avait	$(o/p)/q$
ni	$q/r, p/s$	écrit	r	cet	n/t
accord	u	dont	$(u\backslash t)/v$	hérité	x
.	$k\backslash j, a\backslash b$				

TABLE 2 – Lexique avant unification

Entre ces algorithmes standards d'inférence grammaticale et le nôtre, il y a deux différences principales.

La première différence réside dans le fait que nous utilisons les informations provenant des annotations du corpus, comme résumé dans la table 3. Les types assignés aux noeuds ont été choisis en fonction de leur fréquence dans le lexique extrait des arbres après transduction. Si le label n'est pas dans la table, le type du noeud sera une variable dans le cas d'un noeud argument. Si le noeud est foncteur, son type sera instancié en même temps que celui de son argument, puisque cette méthode est descendante. Les arbres utilisés en entrée contiennent donc des sous-formules avec des variables libres. L'inférence grammaticale consiste à transformer ces arbres aux types partiellement spécifiés en arbres de dérivation sans variable. Un exemple d'arbre d'entrée est montré figure 4. On remarque que certains mots, même si leur POS-tag n'est pas dans la table 3, ont déjà des types complexes sans variable.

label	**type**	**label**	**type**
TEXT	txt	SENT	s
NP	np	NP-ARG	np
PP	pp	AP-ARG	n\n
CLS	np	CLS-SUJ	np
NC	n	NPP	np
VPP	np\ s_p	VINF	np \ s_i

TABLE 3 – Extrait de la liste des types assignés aux noeuds lorsque ceux-ci ont le bon label, si et seulement s'ils sont arguments et non foncteurs au niveau des dérivations d'une grammaire AB.

La seconde différence est l'utilisation de clusters pour guider l'étape d'unification. Nous avons pris le parti d'utiliser un algorithme de clustering hiérarchique pour ce faire. Un cluster peut aussi bien regrouper un ensemble de cluster que des mots. Chaque cluster est associé à une hauteur qui représente la similarité entre les données qu'il regroupe. Ainsi, les clusters de hauteur zéro regroupent les mots dont les vecteurs sont identiques, et les clusters de hauteur plus importante regroupent aussi bien des mots que d'autres clusters qui leurs sont proches, comme montré figure 5. Nous unifions les clusters par hauteur croissante, ce qui nous permet d'unifier par ordre de

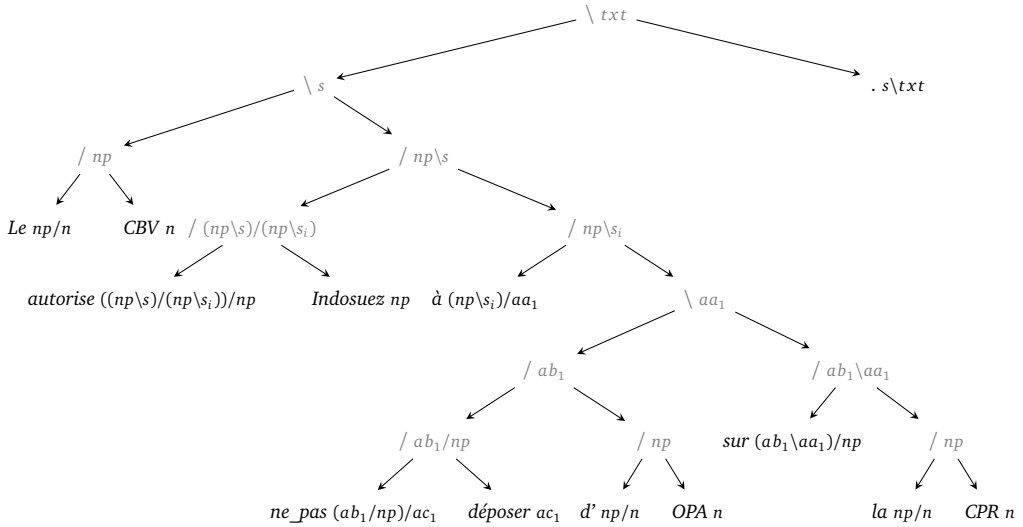

FIGURE 4 – Arbre d'entrée pour notre algorithme d'inférence grammaticale. Certains types sont déjà traités par l'étape de transduction et les autres sont remplacés par des variables.

similarité. Lorsque, pour une hauteur donnée, nous avons plusieurs possibilités d'unification, nous appliquons un ordre de priorité qui peut être résumé par :

1. unifier les plus petits clusters,
2. unifier les clusters pour lesquels il n'y a qu'un seul choix par variable,
3. unifier avec le plus simple candidat (la complexité d'un candidat est calculée en fonction du nombre de \et de / qu'il contient),
4. choisir le premier venu pour les autres variables, avec la possibilité de choisir aléatoirement l'unification.

Il se peut que tous les mots ne soient pas représentés à un niveau donné, par conséquent il peut rester des variables dans les types après une étape de clustering. Dans ce cas, on passe à un nouveau niveau de clustering. Cette manière de procéder nous assure l'unification des variables qui apparaissent en premier lieu dans des contextes les plus similaires possibles.

Il est à noter que même avec des variables, les arbres de dérivation partiels restent des dérivations valides et représentatives d'une grammaire AB : Les deux règles d'élimination sont les seules utilisées pour créer les arbres.

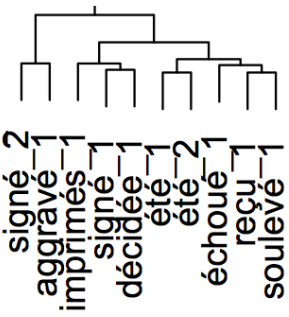

FIGURE 5 – Extrait d'un cluster de 5 phrases. Les participes passés sont regroupés d'abord par contexte puis tous ensemble dans de plus larges clusters jusqu'à n'en former plus qu'un.

4 Clustering

4.1 Extraction de vecteurs

Nous avons décidé d'assigner des vecteurs aux mots en extrayant les informations des corpus avant transductions.

Les vecteurs ont six dimensions :

1 POS-tag du mot (père),

2 information morpho-syntaxique (grand-père),

3-4 POS-tag du frère à gauche et à droite,

5-6 distance jusqu'au plus proche ancêtre commun avec le voisin de gauche et de droite.

S'il n'y a pas de voisin de droite ou de gauche (dernier ou premier mot d'une phrase), la valeur correspondant à la coordonnée de ce vecteur sera instanciée à NIL ou -5, suivant si c'est un label ou un nombre. Deux exemples de vecteurs sont donnés figure 6.

$$le_1 < \text{DET, NP-SUJ}, NIL, \text{NC}, -5, 2 >$$

$$le_2 < \text{DET, NP-MOD, VPP, ADJ}, 3, 2 >$$

FIGURE 6 – Deux vecteurs correspondant au déterminant "le".

Pour comparer les vecteurs, nous avons besoin de les transformer en vecteurs dans $\mathbb{Z}^n, n \in \mathbb{N}$. Nous avons pris le parti de transformer chaque label en vecteur où seulement une ou deux dimensions possède la valeur 1 et le reste des coordonnées a pour valeur 0. Les POS-tags et les informations syntaxiques sont transformés de cette manière. Les distances numériques restent telles quelles, comme montré figure 6. La transformation est illustrée par la table 4 avec seulement une portion des données. Il y a une "dimension" pour presque chacun des POS-tags (avec cependant quelques exceptions pour des cas que nous souhaitons voir unifiés ensemble,

comme *ET* pour les mots étrangers et *NPP* pour les noms propres) ; pour les informations morpho-syntaxiques, en plus d'une dimension pour chaque catégorie de base (*NP*, *PP*...) on fait seulement la différence entre les arguments (représentés par le *-SUJ, -OBJ, -ATS*... à la fin des labels) et les modificateurs *-MOD*.

POS-tag	**NC**	**DET**	**P**	**...**
NC	1	0	0	0...0
DET	0	1	0	0...0
P+D	0	1	1	0...0
Other	**NP**	**...**	**-ARG**	**-MOD**
NP	1	0...0		
NP-SUJ	1	0...0	1	0
NP-MOD	1	0...0	0	1

TABLE 4 – Exemple de transformation de vecteurs.

4.2 Création des clusters

Pour calculer le cluster hiérarchique nous utilisons le logiciel R (Ihaka et Gentleman, 1993), la distance métrique Manhattan et pour le clustering en lui-même la méthode de variance minimum de Ward (Ward, 1963). Pour mieux visualiser le cluster complet nous utilisons Tulip (Auber et Mary, 2007), ce qui permet de créer des graphes comme ceux de la figure 7. Les détails du graphe seront montrés dans la section suivante.

5 Evaluation

Nous avons testé notre méthode sur 754 phrase de Sequoia et 553 phrases du corpus de Paris VII. Le tableau 5 montre l'efficacité de la méthode, calculée en fonction du pourcentage de variables restant après unification.

Corpus	Sequoia (754 phrases)	Paris VII (553 phrases)
variables restantes	3	0
nombre total de variables	1.429	686
ratio	99,8%	100%

TABLE 5 – Pourcentage de variables restantes après unifications sur l'extrait de Sequoia et Paris VII.

Cependant le nombre de variables restantes sont un faible critère de succès. En effet, si les variables sont unifiées mais que les types résultants sont trop complexes, on ne peut pas dire que notre méthode soit un réel succès, même si toutes les phrases ont un arbre de dérivation valide.

Pour le corpus Sequoia, lorsque l'on compare les lexiques extraits après transduction et avec notre nouvel algorithme d'inférence grammaticale, on peut noter que 82,7% des lexiques sont identiques : cela signifie qu'il y a seulement 2967 paires mot-type différentes sur les 17110

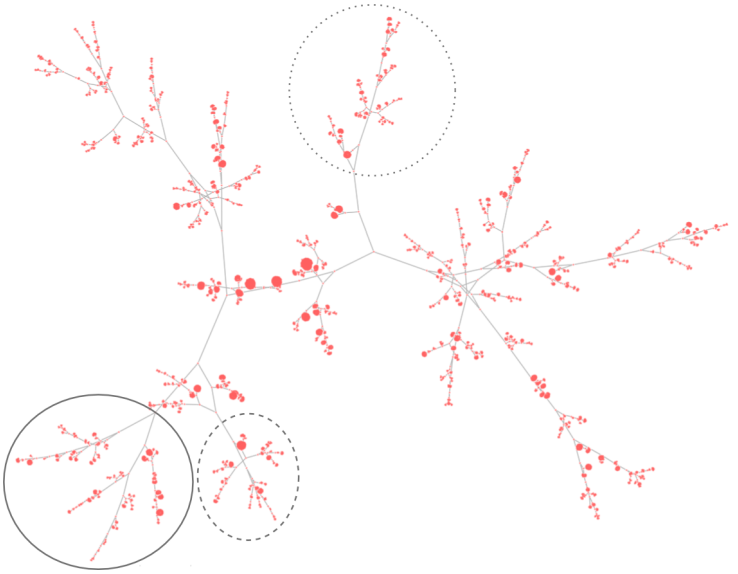

FIGURE 7 – Cluster correspondant à notre ensemble test de phrases. La partie entourée en pointillés correspond à l'endroit où il y a le plus de verbes ; celle entourée par des tirets aux déterminants et la partie simplement cerclée correspond aux adjectifs. On peut noter que les adjectifs et les déterminants sont proches les uns des autres. Cela s'explique parce qu'ils prennent généralement tous les deux un nom commun en argument et qu'ils sont présents dans des groupes nominaux.

paires du lexique. Cela correspond à 1012 entrées dans le lexique qui ont au moins une différence.

Pour les 553 phrases du corpus de Paris VII, nous comparons plus en détail les deux lexiques. Cela correspond à 2076 mots, soit 5731 paires mot-types.

Les différences entre les deux lexiques correspondent à 899 paires qui s'étalent sur 379 mots, soit 14,9% du lexique. Cela signifie que 85,1% des lexiques sont identiques. Dans ces 85,1% il faut noter cependant qu'il y a 2% de modifications mineures, telles qu'un *np* qui devient un *n* (majoritairement dans des cas tels que "Le président Merem") ou qu'une inversion entre les différents types des prépositions, pp, pp_{de} ou pp_{a} (les trois correspondent à des syntagmes prépositionnels, mais les deux derniers ajoutent comme information que la préposition utilisée est un *à* ou un *de*. Il faut noter cependant que certaines prépositions ne sont pas annotées comme ayant un *à* ou un *de* dans le corpus). Nous travaillons actuellement à faire disparaître ces modifications mineures.

Le tableau 6 trie les différences en deux catégories : d'un côté les types qui sont présents dans le lexique provenant du transducteur mais qui n'ont pas la même occurrence, et de l'autre ceux qui n'apparaissent pas dans le lexique de référence. Quelques exemples sont montrés dans le tableau 7. Le participe passé *accumulé* peut être utilisé aussi bien comme un adjectif. Le type donné par l'unification correspond à un noeud VPP utilisé comme un participe passé et non comme un

adjectif, ce qui pourtant correspond mieux au contexte, cependant la *CCG Bank* (Hockenmaier et Steedman, 2007) contient une règle spéciale qui permet une translation de $np\backslash s_p$ à $n\backslash n$; on peut donc dire que le fait de considérer les deux types comme équivalents pour l'évaluation semble être justifié.

Le type donné à *change* est une vraie erreur : à la place d'être traité comme un verbe transitif qui prendrait donc deux arguments, il est traité comme un verbe intransitif. Cette erreur vient de l'étape de clustering, où *change* est proche d'un autre verbe intransitif.

paires erronées	569	8,7%
paires équivalentes	336	6,2%
paires identiques	4 832	85,1%
paires utilisables	5 168	**91,3%**

TABLE 6 – Ratio entre les différences des lexiques, comptées en paire mot-type. On note que 91,3% du lexique est sans erreur, donc utilisable en l'état.

Mot	**Unification**	**Transduction**
accumulé	$np\backslash s_p$	$n\backslash n$
change	$np\backslash s$	$(np\backslash s)/np$

TABLE 7 – Un exemple de chaque classe de mots.

La figure 8 montre deux clusters de niveau zéro. Le gauche est un cluster d'adverbes. La variable ab_{331} sera unifiée avec np. Le cluster de droite contient uniquement des variables qui seront unifiées en une seule. Il rassemble des adjectifs et des participes passés utilisés en tant qu'adjectifs.

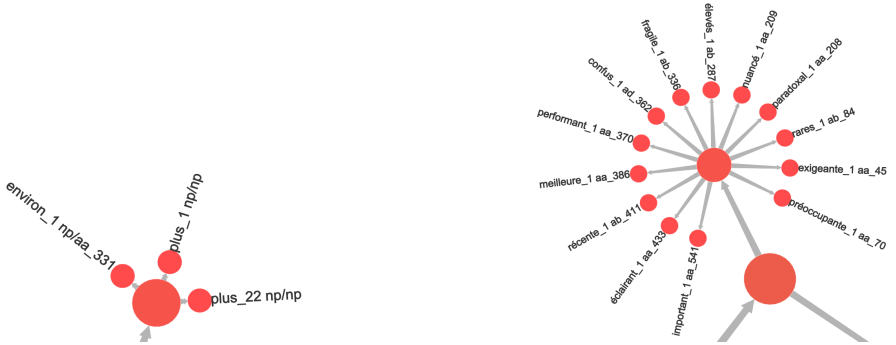

FIGURE 8 – Zoom sur deux clusters de niveau zéro.

6 Discussions

La méthode que nous utilisons est faite pour fonctionner avec des arbres de dérivation : il s'agit d'apprentissage non supervisé, mais avec des structures d'entrée contenant beaucoup d'informations dont des informations syntaxiques. Cependant, cela pourrait être étendu à n'importe

quel ensemble de phrases qui ne sont pas sous forme d'arbres avec quelques modifications. L'idée serait d'utiliser à la fois des phrases simples et d'autres phrases sous forme d'arbres de dérivation.

Le problème est d'avoir les vecteurs de mots pour les phrases qui n'ont pas d'arbres syntaxiques attachés. On pourrait alors utiliser les vecteurs d'un sous-espace car certaines informations, comme le POS-tag des mots, peuvent être facilement retrouvées avec un tagger (Moot, 2012).

Ensuite, nous pouvons effectuer une étape de clustering avec ces vecteurs partiels et ceux extraits d'arbres syntaxiques en faisant une projection sur les seconds pour diminuer le nombre de dimensions. De cette manière, les mots pourraient avoir le type le plus utilisé par le cluster de niveau zéro leur correspondant. Cela nous permettrait d'avoir une plus grande visibilité sur les mots que si nous leur donnions juste le type le plus utilisé dans un lexique de référence en fonction de leur POS-tag. Etant donné que nos vecteurs ont un grand nombre de dimensions et sont très vides, nous pourrions aussi appliquer la méthode décrite par Kailing et al. (Kailing *et al.*, 2004) pour les manipuler.

6.1 Application à de plus grands corpus

Nous souhaitons appliquer notre méthode actuelle à des ensembles plus larges, mais nous aurons alors affaire à des clusters beaucoup plus larges pour le corpus Sequoia complet (plus de 63000 mots) ou encore pour le corpus de Paris VII (environ 300000 mots). L'étape de clustering est, avec la méthode de Ward (Ward, 1963), d'une complexité $O(n^3)$, et cela commence à devenir problématique pour ces grands ensembles. Il faut cependant noter que cela constitue une amélioration par rapport aux autres algorithmes d'apprentissage, étant donné que les grammaires k-valuées ont une complexité exponentielle.

6.2 Optimisation de l'unification des types

Pour l'instant, nous utilisons le critère "premier trouvé" pour unifier les variables lorsque nous n'avons pas d'autre critère de choix. Une solution plus optimale serait de regarder toutes les variables dans leur globalité, de leur assigner une liste d'unifications possibles et d'utiliser l'algorithme de Kuhn-Munkres (Kuhn, 1955; Munkres, 1957) pour choisir la meilleure unification globale, comme par exemple celle qui donne l'instantiation des variables avec les types les plus simples.

7 Conclusion

Dans cet article, nous avons montré une nouvelle méthode pour extraire une grammaire AB par unification en utilisant le clustering pour nous guider. Une implémentation est disponible (Sandillon-Rezer, 2013).

Nous avons décidé d'utiliser un clustering hiérarchique, ce qui nous permettait d'unifier le lexique pas à pas, soit jusqu'à convergence, soit jusqu'à ce qu'un conflit bloque l'unification. Cependant, il serait intéressant de tester notre méthode avec d'autres types de clustering, comme la méthode des *k-means* ou celle appelée *Clustering By Committee* (Pantel, 2003). Cette dernière méthode

cherche le meilleur centroïde de chaque cluster qui est sensé être représentatif de chacun ; elle ne peut cependant pas être appliqué en l'état, parce que nous souhaitons faire l'unification après le clustering et que les types des centroïdes ne sont donc pas encore définis.

Les résultats que nous avons sont prometteurs surtout si on garde à l'esprit le fait que le format d'entrée est peu détaillé, ce qui nécessite donc moins d'heures aux annotateurs, mais que nous sommes proches de notre lexique de référence : nous avons 91,3% du lexique qui est similaire et 85,1% qui est identique.

Références

Abeillé, A. et Clément, L. (2003). Annotation morpho-syntaxique.

Abeillé, A., Clément, L. et Toussenel, F. (2003). Building a treebank for french. *Treebanks, Kluwer, Dordrecht*.

Adriaans, P. W. (1999). Learning shallow Context-Free languages under simple distributions.

Ajdukiewicz, K. (1935). Die syntaktische konnexität. *Stud. Philos.*, 1:1–27.

Auber, D. et Mary, P. (2007). Tulip : Better visualization through research.

Bar-Hillel, Y. (1964). *Language and information : selected essays on their theory and application*. Addison-Wesley Pub. Co.

Buszkowski, W. et Penn, G. (1990). Categorial grammars determined from linguistic data by unification. *Studia Logica*.

Candito, M. et Seddah, D. (2012). Le corpussequoia : annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical.

Costa-Florêncio, C. (2001). Consistent identification in the limit of any of the classes k-valued is np-hard. *Lecture Notes in Artificial Intelligence*.

Gold, E. (1967). Language identification in the limit. *Information and Control*, 10.

Hockenmaier, J. (2003). Data and models for statistical parsing with combinatory categorial grammar.

Hockenmaier, J. et Steedman, M. (2007). CCGbank : a corpus of CCG derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396.

Ihaka, R. et Gentleman, R. (1993). R project.

Kailing, K., Kriegel, H. et Kröger, P. (2004). Density-connected subspace clustering for high-dimensional data. *Proceedings of the Fourth SIAM International Conference on Data Mining*.

Kanazawa, M. (1998). *Learnable Classes of Categorial Grammars*. Center for the Study of Language and Information, Stanford University.

Kuhn, H. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*.

Lambek, J. (1958). The mathematics of sentence structure. *The American Mathematical Monthly*, 65.

Moot, R. (2012). Wide-coverage semantics for spatio-temporal reasoning. *Traitement Automatique des Langues 52*.

MUNKRES, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*.

PANTEL, P. (2003). Clustering by committee. *PhD thesis*.

SANDILLON-REZER, N.-F. (2013). http ://www.labri.fr/perso/nfsr.

SANDILLON-REZER, N.-F. et MOOT, R. (2011). Using tree tranducers for grammatical inference. *Proceedings of Logical Aspects of Computational Linguistics 2011*.

TRAUTWEIN, H., ADRIAANS, P. et VERVOORT, M. (2000). Towards high speed grammar induction on large text corpora.

WARD, J. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*.