Un analyseur discriminant de la famille LR pour l'analyse en constituants

Benoît Crabbé
ALPAGE, INRIA, Université Paris Diderot
Place Paul Ricoeur , 75013 Paris
bcrabbe@linguist.univ-paris-diderot.fr

Résumé. On propose un algorithme original d'analyse syntaxique déterministe en constituants pour le langage naturel inspiré de LR (Knuth, 1965). L'algorithme s'appuie sur un modèle d'apprentissage discriminant pour réaliser la désambiguisation (Collins, 2002). On montre que le modèle discriminant permet de capturer plus finement de l'information morphologique présente dans les données, ce qui lui permet d'obtenir des résultats état de l'art en temps comme en exactitude pour l'analyse syntaxique du français.

Abstract. We provide a new weighted parsing algorithm for deterministic context free grammar parsing inspired by LR (Knuth, 1965). The parser is weighted by a discriminative model that allows determinism (Collins, 2002). We show that the discriminative model allows to take advantage of morphological information available in the data, hence allowing to achieve state of the art results both in time and in accurracy for parsing French.

Mots-clés : Analyse guidée par les têtes, analyse LR, temps linéaire, modèle discriminant,inférence approximative.

Keywords: Head driven parsing, LR parsing, linear time, discriminative modelling, approximate inference.

1 Introduction

Cet article présente un algorithme d'analyse syntaxique robuste inspiré des algorithmes LR (Knuth, 1965) et GLR (Tomita, 1985) pour les grammaires de réécritures non contextuelles. L'algorithme est augmenté d'un mécanisme de pondération permettant la désambiguisation. Celui-ci est basé sur l'algorithme du perceptron global (Collins, 2002). L'article montre que cet algorithme est état de l'art en temps comme en correction sur un jeu de données de référence pour le français (Abeillé *et al.*, 2003; Seddah *et al.*, 2013).

L'analyse syntaxique en constituants repose sur l'hypothèse que caractériser la structure d'une phrase de la langue en modélisant la manière dont les mots sont groupés a du sens. De plus il s'agit d'une représentation particulièrement adaptée à la construction compositionnelle du sens des phrases, comme illustré par (Socher *et al.*, 2012). Il reste que les principaux modèles d'analyse probabiliste en constituants sont génératifs et ont été conçus en priorité pour l'analyse de l'anglais (Petrov *et al.*, 2006; Charniak, 2000; Collins, 2003) ou le chinois (Zhang, 2009), c'est-à-dire des langues à morphologie très pauvre. Pour l'analyse de langues à morphologie plus riche, comme le français, on fait l'hypothèse qu'il est souhaitable de pouvoir se donner une représentation structurée des mots (lemmatisation, analyse morphologique, représentation sémantique) issue directement de treebanks voire de ressources exogènes, comme des dictionnaires. Et ce, dans le but de capturer certaines de leurs propriétés essentielles et de combattre les effets de dispersion des données. Il est cependant non trivial d'adapter les modèles génératifs développés en priorité pour l'anglais au cas des langues à morphologie riche. Il faudrait notamment mettre en place une factorisation du modèle génératif relativement lourde impliquant notamment le déploiement de méthodes de lissage très élaborées. Pour cette raison, il semble préférable d'utiliser un modèle d'apprentissage discriminant qui offre naturellement la possibilité de factoriser le modèle d'analyse.

Cependant, comme montré par (Finkel *et al.*, 2008) la conception de modèles d'analyse syntaxique en constituants entièrement discriminants est une tâche difficile qui pose des problèmes notoires d'efficacité. Ainsi la pratique courante pour l'analyse syntaxique en constituants consiste plutôt à utiliser un modèle discriminant pour réordonnancer un sous-ensemble des hypothèses d'analyse construites par un analyseur génératif efficace (Charniak & Johnson, 2005). Dans cet article on montre qu'en introduisant une approximation appropriée, on peut formuler directement un algorithme d'analyse en constituants basé sur un modèle discriminant qui est plus expressif et plus efficace que ses contreparties génératives. On

montre en particulier qu'il permet de tirer parti de l'information morphologique naturellement présente dans les données pour obtenir des résultats état de l'art pour le français en temps comme en correction.

L'article est structuré comme suit. On commence par établir en section 2 un cadre formel approprié qui permet notamment de contraster le problème de l'analyse en constituants avec celui de l'analyse en dépendances. Ayant observé que la structure des arbres de constituants est contrainte, on propose en section 3 une méthode destinée à contraindre l'espace des analyses possibles par un automate LR pour le cas de l'analyse robuste, ce qui distingue notre proposition de (Sagae & Lavie, 2006; Zhang, 2009; Zhu *et al.*, 2013). La section 4 propose un algorithme d'analyse de la famille LR et un algorithme d'apprentissage basé sur le modèle du perceptron global. On y détaille en particulier l'usage d'une méthode d'inférence approximative basée sur un faisceau et ses conséquences sur la méthode d'apprentissage. Les sections 5 et 6 introduisent enfin les extensions de l'algorithme de base qui sont utiles en pratique et qui permettent de structurer les formes lexicales. Finalement la section 7 donne une évaluation quantitative des différents composants de l'algorithme et une comparaison avec l'état de l'art.

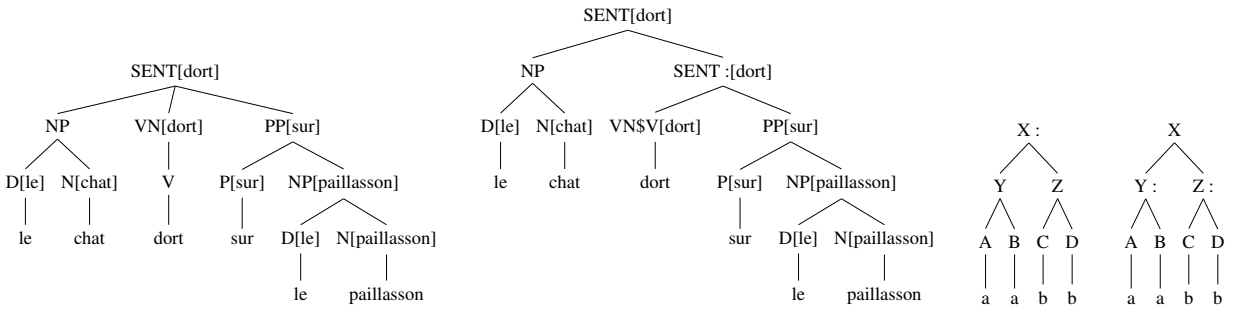
2 Représentation de la grammaire

On pose comme hypothèse de départ que la grammaire manipulée lors de l'analyse est une grammaire lexicalisée de type 2-LCFG (Nederhof & Satta, 2010). Une grammaire 2-LCFG est une grammaire CFG dont les règles ont nécessairement la forme donnée en Table 1 où les symboles h, x dénotent des symboles terminaux. Les symboles de la forme $A[h]$ dénotent des symboles non terminaux lexicalisés. Un tel symbole est composé d'un non terminal délexicalisé noté A, B, C et d'un terminal h ou x. Une règle 2-LCFG sera par exemple de la forme $NP[chat] \rightarrow D[le]\ N[chat]$. Le symbole de la forme $X[h]$ situé en partie droite de la règle est appelé tête de la règle. Une grammaire de type 2-LCFG comporte en pratique un très grand nombre de règles. L'analyse efficace de ce type de grammaire demande typiquement de générer dynamiquement les non terminaux lexicalisés en cours d'analyse. Dans ce qui suit, nous focalisons d'abord sur les propriétés de la grammaire 2-CFG sous jacente. La partie dynamique sera détaillée en Section 4.

$$\begin{array}{lcl} A[h] & \rightarrow & B[h]\,C[x] \\ A[h] & \rightarrow & B[x]\,C[h] \\ A[h] & \rightarrow & h \end{array}$$

TABLE 1 – Formes des règles 2-LCFG

Transformation du treebank Le treebank français, comme la plupart des treebanks existants encode des arbres d'arité n, avec notamment $n > 2$. En supposant un tel treebank, dont les noeuds sont tous annotés par leurs têtes, on effectue les opérations de précompilation suivantes : d'abord une opération de binarisation, appelée Markovisation par la tête d'ordre 0 (Collins, 2003), suivie d'une procédure de réduction des règles unaires internes, comme illustré en Figure 1. Ces deux opérations garantissent que le treebank transformé comporte uniquement des arbres dont la structure suit une forme normale de Chomsky. La procédure de binarisation introduit un ensemble de catégories temporaires que nous notons avec le suffixe ':' en Figure 1. Le symbole '$' dénote les catégories introduites par la réduction de règles unaires.



Arbre source — Arbre 2-LCFG transformé — Arbres 2-CFG invalides

FIGURE 1 – Représentation des arbres de treebank par une 2-LCFG

Propriétés des arbres transformés Comme le treebank est mis en forme normale de Chomsky, on peut montrer par induction que le nombre de pas de dérivation η pour dériver à partir de la grammaire 2-CFG sous-jacente au treebank

tout arbre d'analyse d'une phrase de n mots est constant : $\eta = 2n - 1$. Cette propriété est en principe essentielle (voir également section 5) et explique pourquoi on utilise 2-LCFG pour représenter la grammaire. On remarque de plus que les arbres du treebank transformé ont une structure contrainte. Par exemple la racine d'un arbre ne peut pas être un symbole temporaire ou un arbre ne peut pas avoir deux noeuds temporaires en partie droite d'une même règle. Les analyseurs en dépendances projectifs vérifient également la première propriété : $\eta = 2n - 1$ (Huang & Sagae, 2010). Par contre les structures à analyser ne sont pas contraintes de manière identique.

3 Construction d'un automate LR pour une grammaire de treebank

Les propriétés des arbres transformés, induites par l'introduction des symboles temporaires, ont pour conséquence qu'on ne peut pas construire naïvement un analyseur par décalage réduction non contraint comme c'est le cas pour l'analyse en dépendances. Dans le cas 2-CFG présenté ici, les arbres d'analyse ont une structure contrainte et il s'agit de garantir que l'analyseur produise des arbres qui respectent ces contraintes. Dans son parser, (Sagae & Lavie, 2006) ne gère pas ce problème et retourne à posteriori des analyse partielles dans le cas où l'arbre d'analyse s'avère invalide. (Zhang, 2009) exprime plutôt des contraintes locales dans l'algorithme d'analyse qui sont destinées à empêcher l'analyseur de produire des arbres invalides. Celles-ci comportent notamment des contraintes d'ordre général et des contraintes plus spécifiques à la grammaire du chinois manipulée par l'auteur. On propose ici une solution plus générale qui consiste à garantir la correction des arbres d'analyse par l'utilisation de tables $LR(0)$. Tel quel, un analyseur $LR(0)$ traditionnel (Knuth, 1965) n'est pas approprié pour l'analyse du langage naturel : le principe de cette méthode d'analyse est d'éliminer statiquement l'ambiguité de la grammaire. Dans ce qui suit, à l'instar de (Tomita, 1985) on construit des tables LR sans chercher à éliminer statiquement l'ambiguité. Au contraire, on préserve les conflits dans la table : la désambiguisation est réalisée dynamiquement par un mécanisme de pondération. L'utilisation de tables $LR(0)$ permet plutôt de garantir que l'analyseur produit des arbres corrects et dérivés d'une grammaire bien identifiée. Contraindre la structure des arbres d'analyse revient alors à contraindre la grammaire ayant servi à générer ces tables. Ceci dit, construire un automate $LR(0)$ pour l'analyse robuste à partir de treebank pose à priori deux problèmes. Le premier est inductif : il faut garantir que la grammaire extraite d'un treebank généralise à du texte non vu. Rien ne dit que la grammaire issue d'un échantillon limité est générale. Le second est d'ordre pratique : une grammaire extraite de treebank est généralement une grammaire de très grande taille et massivement ambigue. Cette seconde propriété rend la compilation d'automates $LR(0)$ délicate en pratique : la construction de tables $LR(0)$ fait intervenir un algorithme de déterminisation d'automates dont la complexité est en $\mathcal{O}(2^n)$ avec n le nombre d'états de l'automate LR non déterministe. Dans le cas de très grosses grammaires ambigues, n est très élevé et on est proche du pire des cas.

Pour ces deux raisons, et dans le cas où l'analyseur résultant a pour but d'être robuste, on propose de dériver l'automate à partir d'une construction grammaticale basée sur des classes d'équivalence. On commence par poser que Σ est l'ensemble des symboles non terminaux extraits du treebank tel que T est l'ensemble des symboles temporaires introduits par la procédure de binarisation et N est l'ensemble des autres symboles ($\Sigma = N \cup T$ et $N \cap T = \emptyset$). On note W l'ensemble des symboles terminaux extraits du treebank et $A \in \Sigma$ l'axiome unique de cette grammaire ($A \in N$). On définit ensuite un ensemble de classes d'équivalence qui partitionnent Σ : $[a] = \{A\}$, $[t] = T$ et $[n] = \Sigma - (T \cup \{A\})$. Pour uniformiser les notations on définit $[w] = W$. Munis de ces classes d'équivalence, on définit la grammaire matrice $G_m = \langle \Sigma_m, [w], [a], R_m \rangle$ où $\Sigma_m = \{[a], [t], [n]\}$. Reste à définir R_m pour que les règles respectent les contraintes de bonne formation des arbres. On formule en table 2 un exemple de telles règles au format ID/LP (Gazdar *et al.*, 1985). Autrement dit, une règle de dominance immédiate de la forme $a \rightarrow b\,,\ c$ est expansée en deux règles de réécriture $a \rightarrow b\,c$ et $a \rightarrow c\,b$.

$[a] \rightarrow [n]\,,\ [t]$	$[n] \rightarrow [n]\,,\ [t]$	$[t] \rightarrow [n]\,,\ [t]$
$[a] \rightarrow [n]\,,\ [n]$	$[n] \rightarrow [n]\,,\ [n]$	$[t] \rightarrow [n]\,,\ [n]$
$[a] \rightarrow [w]$	$[n] \rightarrow [w]$	

TABLE 2 – Exemple de règles de dominance immédiate pour G_m

La grammaire G_m implémente les contraintes de bonne formation des arbres mentionnées ci-dessus, comporte en pratique peu de règles et elle est robuste : $L(G_m) = [w]^*$. On peut construire très facilement à partir de G_m un automate $LR(0)$ déterministe $A_m = \langle \Sigma_m \cup \{[w]\}, Q, i, F, E_m \rangle$ en utilisant les méthodes classiques (Aho *et al.*, 2006). On construit l'automate expansé $A_{exp} = \langle \{\Sigma \cup W\}, Q, i, F, E \rangle$ où $E = \{(q, a, q') \,|\, (q, [x], q') \in E_m, \forall a \in [x]\}$. L'automate A_{exp}

peut alors être utilisé comme guide par un analyseur $LR(0)$ non déterministe qui garantit de produire des arbres d'analyse corrects.

Pour construire la table d'analyse LR, il faut encore définir l'ensemble des actions $\mathcal{A}$ de l'analyseur : $\mathcal{A} \stackrel{def}{=} \{RL(X)|X \in \Sigma\} \cup \{RR(X)|X \in \Sigma\} \cup \{RU(X)|X \in \Sigma\} \cup \{S\}$. Il s'agit d'un jeu d'actions analogue à celui de (Sagae & Lavie, 2006). Si les actions sont définies explicitement dans la section suivante, signalons déjà que $RL(X)$ (resp. $RR(X)$) est une action de réduction de règle binaire par le non terminal X tel que le premier (resp. second) élément de la partie droite est assigné comme tête de la règle. $RU(X)$ dénote une réduction unaire par un non terminal X, et S le décalage. Par contraste avec un jeu d'actions LR classique (Aho *et al.*, 2006), celui-ci introduit une approximation : pour un état de l'automate donné, on introduit une action $RL(X)$ et une action $RR(X)$ si il existe un item LR de la forme $\langle X \rightarrow A\, B\bullet \rangle$ dans un état $q_i \in A_{exp}$ sans exiger que $\langle X \rightarrow B\, A\bullet \rangle \in q_i$. Cette simplification permet de réduire le nombre d'actions de l'analyseur, ce qui facilite l'apprentissage du modèle et a également un effet optimisant (Section 4).

Si la grammaire matrice G_m présentée ici n'est pas la seule grammaire matrice possible (voir également Figure 7 pour l'expression de sous-grammaires locales), un jeu de règles R_m valide pour G_m doit au moins implémenter les contraintes de bonne formation des arbres mentionnées ci-dessus en s'appuyant sur une partition de Σ en classes d'équivalence. Par contre le jeu d'actions simplifié utilisé ici suppose que pour toute règle de grammaire $R \in R_m$ de la forme $[x] \rightarrow [y]\,[z]$, on a également une règle de grammaire $R' \in R_m$ de la forme $[x] \rightarrow [z]\,[y]$. C'est pour cette raison que nous formulons les règles R_m au format ID/LP et cela signifie qu'on ne peut imposer de contraintes dures sur l'ordre des mots dans la grammaire, comme c'est le cas dans la plupart des analyseurs syntaxiques robustes.

4 Algorithme d'analyse inspiré de LR et pondéré par un perceptron

Bien qu'inspiré de LR, l'algorithme d'analyse proposé est un algorithme naturellement non déterministe. Le déterminisme est apporté par un système de pondérations basé sur l'algorithme du perceptron global (Collins, 2002). On commence par présenter l'algorithme d'analyse pondéré avant de décrire la méthode d'estimation des poids du perceptron à partir d'un treebank.

Algorithme d'analyse On suppose que l'algorithme analyse des séquences de tokens $\mathcal{T} = t_1 \ldots t_n$ et qu'une table d'analyse $LR(0)$ a été construite. La fonction GOTO de cette table est la fonction GOTO :$(\Sigma \cup W) \times \mathbb{N} \mapsto \mathbb{N}$ qui envoie des couples de symbole et d'état vers un nouvel état de l'automate LR. La fonction ACTION de cette table est la fonction ACTION :$(\mathbb{N} \times W) \mapsto 2^{\mathcal{A}}$ qui retourne l'ensemble **a** des actions possibles étant donné un couple de terminal et de numéro d'état. On note σ_i l'état initial de l'automate LR et σ_e un état final.

Les algorithmes à décalage réduction manipulent habituellement une pile et une liste d'attente. Ici, l'algorithme manipule explicitement une pile et implicitement une liste d'attente. La pile $\mathbf{S} = \ldots|s_2|s_1|s_0$, de sommet s_0, est faite de noeuds de la forme $s_i = \langle \sigma, \tau \rangle$ où σ est un numéro d'état LR et $\tau = (s_i.c_t[s_i.w_t]\; s_i.c_l[s_i.w_l]\; s_i.c_r[s_i.w_r])$ dénote un arbre local de profondeur 1. $s_i.c_t, s_i.c_l, s_i.c_r$ dénotent respectivement les catégories du noeud racine, de son fils gauche et de son fils droit. $s_i.w_t, s_i.w_l, s_i.w_r$ dénotent respectivement les items lexicaux du noeud racine, de son fils gauche et de son fils droit. La notation $s_i.c.[s_i.w.]$ encode donc un symbole non terminal d'une 2-LCFG au noeud s_i de la pile d'analyse (Figure 2).

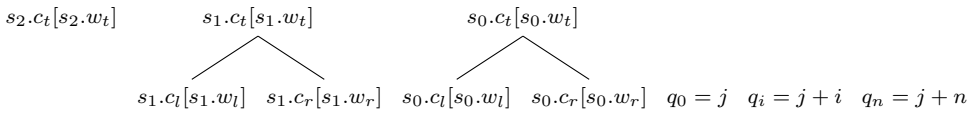


FIGURE 2 – *Représentation graphique du vecteur d'accroches* $\boldsymbol{\kappa}$ Le vecteur $\boldsymbol{\kappa}$ représente l'information localement accessible pour valuer les traits $\phi_i(a, \boldsymbol{\kappa}, j)$. On représente graphiquement de droite à gauche les différents noeuds de la pile d'exécution. Les symboles $s_i.w_{(.)}$ encodent les formes lexicales têtes des constituants et les symboles $s_i.c_{(.)}$ encodent les catégories des constituants. De plus, les symboles q_i encodent les mots de la file d'attente. On remarque que connaître la valeur de l'index j permet d'addresser non seulement le premier mot de la file d'attente mais également les suivants.

L'algorithme d'analyse fonctionne en empilant et en dépilant des noeuds de la pile **S** et en défilant progressivement la file d'attente. Ainsi la configuration courante ou état de l'analyseur est un couple $C_i = \langle j, \mathbf{S} \rangle$ qui dénote la position courante

ITEM	$\langle j, \mathbf{S}\rangle : w$
INIT	$\langle 1, \langle \sigma_i, \epsilon\rangle\rangle : 0, \emptyset$
GOAL	$\langle n+1, \langle \sigma_e, \mathbf{S}\rangle\rangle : w$

SHIFT $\dfrac{\langle j,\mathbf{S}_\ominus \mid s_0{=}\langle \sigma,\_\rangle\rangle : w}{\langle j{+}1,\mathbf{S}_\ominus \mid s_0 \mid \langle \textsc{Goto}(t_j,\sigma),(t_j[j]\_\_)\rangle : w{+}F(S,\langle j,\mathbf{S}\rangle)}$

RL(X) $\dfrac{\langle j,\mathbf{S}_\ominus \mid s_2{=}\langle \sigma_2,\_\rangle{:}w_2 \mid s_1{=}\langle \sigma_1,(s_1.c_t[s_1.w_t]\_\_)\rangle{:}w_1 \mid s_0{=}\langle \sigma_0,(s_0.c_t[s_0.w_t]\_\_)\rangle\rangle{:}w_0}{\langle j,\mathbf{S}_\ominus \mid s_2 \mid \langle \textsc{Goto}(X,\sigma_2),(X[s_1.w_t]\ s_1.c_t[s_1.w_t]\ s_0.c_t[s_0.w_t])\rangle{:}w_0{+}F(RL(X),\langle j,\mathbf{S}\rangle)}$

RR(X) $\dfrac{\langle j,\mathbf{S}_\ominus \mid s_2{=}\langle \sigma_2,\_\rangle{:}w_2 \mid s_1{=}\langle \sigma_1,(s_1.c_t[s_1.w_t]\_\_)\rangle{:}w_1 \mid s_0{=}\langle \sigma_0,(s_0.c_t[s_0.w_t]\_\_)\rangle\rangle{:}w_0}{\langle j,\mathbf{S}_\ominus \mid s_2 \mid \langle \textsc{Goto}(X,\sigma_2),(X[s_0.w_t]\ s_1.c_t[s_1.w_t]\ s_0.c_t[s_0.w_t])\rangle{:}w_0{+}F(RR(X),\langle j,\mathbf{S}\rangle)}$

RU(X) $\dfrac{\langle j,\mathbf{S}_\ominus \mid s_1{=}\langle \sigma_1,(s_1.c_t[s_1.w_t]\_\_)\rangle \mid s_0{=}\langle \sigma_0,(s_0.c_t[s_0.w_t]\_\_)\rangle\rangle{:}w_0}{\langle j,\mathbf{S}_\ominus \mid s_1 \mid \langle \textsc{Goto}(X,\sigma_1),(X[s_0.w_t]\ s_0.c_t[s_0.w_t])\rangle{:}w_0{+}F(RU(X),\langle j,\mathbf{S}\rangle)}$

GR $\dfrac{\langle j,\mathbf{S}_\ominus \mid s_1{=}\langle \sigma_1,(s_1.c_t[s_1.w_t]\_\_)\rangle \mid s_0{=}\langle \sigma_0,(s_0.c_t[s_0.w_t]\_\_)\rangle\rangle{:}w_0}{\langle j,\mathbf{S}_\ominus \mid s_1 \mid \langle \textsc{Goto}(GR,\sigma_1),(s_0.c_t[s_0.w_t]\_\_)\rangle{:}w_0{+}F(GR,\langle j,\mathbf{S}\rangle)}$ (Règle introduite en section 5)

FIGURE 3 – Règles d'inférence de l'algorithme en notation déductive étendue

dans la file d'attente et l'état courant de la pile. Étant donnée une configuration initiale $C_0 = \langle 1, \langle \sigma_i, \epsilon\rangle\rangle$, l'analyseur dérive pas à pas de nouvelles configurations $C_i = \langle j', \mathbf{S}'\rangle$ à partir de configurations $C_{i-1} = \langle j, \mathbf{S}_\ominus | \langle \sigma, \tau\rangle\rangle$ en exécutant une action $a_{i-1} \in \textsc{Action}(\sigma, t_j)$, ce que l'on note $C_{i-1} \overset{a_{i-1}}{\Rightarrow} C_i$. Une dérivation de $k-$pas est la séquence $C_{0\Rightarrow k}$ telle que $C_0 \overset{a_0}{\Rightarrow} \ldots \overset{a_{k-1}}{\Rightarrow} C_k$. Une dérivation est terminée dans deux cas. La configuration $C_{3n-1} = \langle n+1, \langle \sigma, \tau\rangle\rangle$ est générée. Si $\sigma = \sigma_e$ la dérivation est un succès. Une dérivation est également terminée lorsque $\textsc{Action}(\sigma, t_j) = \emptyset$ pour une configuration $C_k = \langle j, \mathbf{S}_\ominus | \langle \sigma, \tau\rangle\rangle$ donnée, c'est le cas d'échec. Les actions exécutées en cours de dérivation modifient la pile d'analyse et l'état d'avancement j sur la liste d'attente. Celles-ci sont définies en Figure 3 en notation déductive étendue.

Une dérivation $C_{0\Rightarrow k} = C_0 \overset{a_0}{\Rightarrow} \ldots \overset{a_{k-1}}{\Rightarrow} C_k$ est également pondérée par une fonction de la forme :

$$W(C_{0\Rightarrow k}) = \mathbf{w} \cdot \mathbf{\Phi}_g(C_{0\Rightarrow k}) = \sum_{i=0}^{k-1} \mathbf{w} \cdot \mathbf{\Phi}(a_i, C_i) = \sum_{i=0}^{k-1} F(a_i, C_i) \tag{1}$$

où $\mathbf{w} \in \mathbb{R}^d$ est un vecteur d-dimensionnel de poids et $\mathbf{\Phi}(a_i, C_i) \in \{0,1\}^d$ est un vecteur d-dimensionnel de traits. Chaque dimension $0 \leq i \leq d$ du vecteur de traits est valuée par une fonction ϕ_i de signature $\phi_i(a, \boldsymbol{\kappa}, j)$ où $\boldsymbol{\kappa}$ est une séquence d'accroches extraite localement parmi les trois éléments supérieurs de la pile comme illustré en Figure 2. Comme la fonction de pondération est décomposée en une somme de termes correspondant aux étapes de la dérivation, le calcul des poids est réalisé dynamiquement en cours d'analyse et est associé à une configuration (Figure 3) de telle sorte que dans le cas pondéré une configuration a la forme étendue $C_k = \langle j, \mathbf{S}\rangle : w$ où $w = W(C_{0\Rightarrow k})$ est le score préfixe de $C_{0\Rightarrow k}$.

Le système de poids permet de modéliser la désambiguisation en autorisant de choisir l'analyse de poids le plus élevé parmi l'ensemble des analyses possibles pour une séquence $\mathcal{T} = t_1 \ldots t_n$ de mots. Dans notre cas, le non déterminisme est introduit dans l'algorithme par la fonction $\textsc{Action}(\sigma, t_j)$ qui renvoie un ensemble $\mathbf{a} \subseteq \mathcal{A}$ d'actions possibles à effectuer étant donné la configuration courante de l'analyseur de telle sorte qu'à partir d'une séquence $C_{0\Rightarrow k-1}$ on peut dériver un ensemble de séquences $\delta(C_{0\Rightarrow k-1}) = \{C_{0\Rightarrow k} | C_{0\Rightarrow k-1} \overset{a_{k-1}}{\Rightarrow} C_{0\Rightarrow k}\}$ à l'étape suivante. Si $\textsc{Gen}_k(\mathcal{T})$ est l'ensemble des dérivations de $k-$pas pour $\mathcal{T}$, trouver la meilleure analyse demande de calculer la solution du problème d'optimisation suivant [1] :

$$\hat{\mathcal{C}} = \underset{C_{0\Rightarrow 3n-1} \in \textsc{Gen}_{3n-1}(\mathcal{T})}{\text{argmax}} W(C_{0\Rightarrow 3n-1}) \tag{2}$$

On peut observer que le nombre de solutions possibles $|\textsc{Gen}_{3n-1}(\mathcal{T})|$ est de l'ordre de $|\mathcal{A}|^{3n-1}$. Autrement dit, l'espace des solutions est de taille exponentielle. Si il est théoriquement possible de calculer en théorie une solution optimale à ce problème par programmation dynamique en temps polynomial dans la lignée de (Tomita, 1985), le mécanisme

1. Notons que le nombre de pas dérivation $\eta = 2n - 1 + n = 3n - 1$ car dans le cas d'analyse par décalage réduction, il faut réaliser n décalages en plus des $2n - 1$ étapes induites par la forme normale de Chomsky.

de pondération utilisé ici demande également d'évaluer des produits scalaires tels que donné en équation (1) pour des vecteurs qui en pratique sont de très haute dimensionnalité. Le calcul dynamique de poids est en pratique très coûteux en temps et rend très difficile la recherche d'une solution à la fois optimale et efficace au problème d'optimisation, même par programmation dynamique. Dans ce qui suit, nous sacrifions la recherche de la solution optimale pour privilégier l'efficacité en faisant appel à une approximation par faisceau. Étant donné un faisceau $\textsc{Gen}^K_{k-1}(\mathcal{T})$ de taille K, et $\Delta(\textsc{Gen}^K_{k-1}(\mathcal{T})) = \bigcup_{C_{0\Rightarrow k-1}\in\textsc{Gen}^K_{k-1}(\mathcal{T})} \delta(C_{0\Rightarrow k-1})$ l'ensemble des configurations dérivables à partir de ce faisceau, on construit récursivement $\textsc{Gen}^K_k(\mathcal{T})$ comme suit :

$$\textsc{Gen}^K_k(\mathcal{T}) = \underset{C_{0\Rightarrow k}\in\Delta(\textsc{Gen}^K_{k-1}(\mathcal{T}))}{\text{K-argmax}} W(C_{0\Rightarrow k}) \tag{3}$$

Autrement dit, un faisceau ne prend en compte que les $K-$meilleures hypothèses de l'étape de calcul précédente pour réaliser les calculs à l'étape courante. Introduire un faisceau apporte un gain d'efficacité qui rend l'algorithme d'analyse utilisable en pratique. La complexité en temps de l'analyse est d'ordre linéaire : $\mathcal{O}(K|\mathcal{A}|(3n-1)) = \mathcal{O}(n)$. L'introduction d'un faisceau a deux contreparties immédiates. D'une part la solution au problème d'analyse donné en équation (2) n'est plus nécessairement optimale. D'autre part on peut montrer que le faisceau sacrifie également la complétude de l'algorithme. Dans certains cas, l'algorithme peut manquer de trouver la moindre analyse alors qu'il est en fait possible d'en trouver au moins une : c'est le cas où l'ensemble des dérivations menant à une solution valide sont tout simplement élaguées prématurément par le faisceau. En résumé, l'algorithme d'analyse présenté ici est une fonction de prédiction d'arbre qui remplace le calcul optimal donné en équation (2) par l'approximation suivante :

$$\tilde{\mathcal{C}} = \underset{C_{0\Rightarrow 3n-1}\in\textsc{Gen}^K_{3n-1}(\mathcal{T})}{\text{argmax}} W(C_{0\Rightarrow 3n-1}) \tag{4}$$

En pratique, le calcul dynamique des poids est le facteur limitant en temps. Trois aspects propres à l'implémentation sont à signaler, notamment pour obtenir un temps d'analyse linéaire en pratique. En premier lieu, l'ensemble des configurations C_k générées en cours d'analyse est implémenté par une structure de données, commune à toutes les hypothèses concurrentes, appelée pile structurée en arbre (*TSS*), initialement introduite par (Tomita, 1988). Cette structure de données permet d'éviter la copie intempestive de piles d'analyse potentiellement profondes en les encodant dans une structure d'arbre.

En second lieu, nous utilisons systématiquement un noyau de hachage (*hash trick*). Un noyau de hachage (Shi *et al.*, 2009) est une technique d'implémentation dont le but premier est d'accélérer l'exécution de produits scalaires. En supposant une fonction de hachage $h : I \mapsto \{1 \dots D\}$ qui envoie l'index $i \in I$ de chaque fonction ϕ_i sur sa valeur hachée $h(i) = k$, de telle sorte que $\bar{\phi}_k(a, \boldsymbol{\kappa}, j) = \phi_i(a, \boldsymbol{\kappa}, j)$. Nous utilisons donc la fonction de pondération suivante plutôt que la première version donnée en équation 1 :

$$F(a_i, C_i) = \mathbf{w} \cdot \bar{\boldsymbol{\Phi}}(a_i, C_i) \tag{5}$$

où $\bar{\boldsymbol{\Phi}} = \bar{\phi}_1 \dots \bar{\phi}_D$ (avec en pratique $D = 2 \times 10^7 - 7$). Autrement dit, l'utilisation d'un noyau de hachage approxime une fonction ϕ_i par sa fonction image $\bar{\phi}_k$ sans chercher à résoudre les collisions éventuelles de hachage. Cette technique prend son sens lorsqu'on sait que les vecteurs $\boldsymbol{\Phi}$ sont des vecteurs creux à très haute dimensionnalité et qu'ils sont habituellement implémentés par des tables de hachage. Or la résolution des collisions dans ces tables est en pratique très coûteuse en temps. Outre le gain en efficacité apporté par cette technique, le noyau de hachage a deux effets de bords utiles : il permet une réduction de la dimensionalité des vecteurs $\boldsymbol{\Phi}$ et $\mathbf{w}$ et il permet dans une certaine mesure de réduire les effets de surentrainement.

En troisième lieu, nous utilisons une méthode de mémoïsation pour l'évaluation des scores. Celle-ci repose sur une décomposition de la fonction F qui remplace la version donnée en équation 5 par sa version finale :

$$F(a_i, C_i) = \mathbf{w} \cdot \boldsymbol{\Phi}(a_i, C_i) + \sum_{m=1}^{M} \boldsymbol{\mu}_m(a_i, C_i) \tag{6}$$

où $\boldsymbol{\mu}_m$ sont des mémo-fonctions. Lorsqu'elle est évaluée la première fois, une mémo fonction mémorise et renvoie la valeur $\mathbf{w} \cdot \boldsymbol{\Phi}_m(a_i, C_i)$. Si elle est évaluée ultérieurement pour un couple (a'_i, C'_i) équivalent, elle renvoie la valeur mémorisée. Le vecteur de traits $\boldsymbol{\Phi}_m = \phi_1 \dots \phi_r$ d'une mémo fonction $\boldsymbol{\mu}_m$ est constitué de fonctions de signatures $\zeta_m(a, \boldsymbol{\kappa}, j)$ identiques. Deux couples $(a_i, C_i), (a'_i, C'_i)$ sont considérés équivalents par la mémo fonction si la valuation qu'ils donnent aux paramètres $a, \boldsymbol{\kappa}, j$ de ζ_m sont identiques. L'analyseur décrit dans cet article utilise deux mémo-fonctions : $\boldsymbol{\mu}_{lex}$ a pour signature $\zeta_{lex}(a, s_0.w_t, s_0.w_l, s_0.w_r, s_1.w_t, s_1.w_l, s_1.w_r, s_2.w_t, j)$ et $\boldsymbol{\mu}_{cat}$ a pour signature $\zeta_{cat}(a, s_0.w_t, s_0.c_t, s_1.w_t, s_1.c_t, j)$. Remarquons également que les traits sont partitionnés dans des ensembles disjoints : chaque mémo-fonction est responsable de l'évaluation d'un sous-ensemble des traits. Les fonctions ϕ_i dont la signature est incompatible avec toutes les mémo-fonctions existantes forment le vecteur $\boldsymbol{\Phi}$ restant (Equation 6) dont le produit scalaire correspondant est réévalué systématiquement.

Estimation des poids La partie prédictive du modèle d'analyse que nous avons décrite jusqu'à présent est en principe valable pour une famille de modèles linéaires multiclasses comme des modèles de type SVM structuré ou des modèles de champs conditionnels aléatoires. La méthode d'estimation des poids décrite ici est spécifique à l'algorithme du perceptron structuré (Collins, 2002). Ce dernier algorithme a en effet l'avantage d'offrir une méthode d'estimation des poids plus efficace et plus simple à mettre en oeuvre pour le cas de l'analyse syntaxique par comparaison avec les autres modèles linéaires que nous connaissons.

On se penche maintenant sur le problème d'estimation du vecteur de poids $\mathbf{w}$ à partir d'un treebank. La méthode d'estimation des poids pour le perceptron structuré est une généralisation de la méthode du percepron multiclasse et se formule comme suit. On suppose un jeu de données d'entrainement de N phrases, $T = ((\mathcal{T}_1, \mathcal{R}_1), \ldots, (\mathcal{T}_N, \mathcal{R}_N))$ pour lesquelles nous disposons de l'analyse correcte $\mathcal{R}_i$. L'algorithme du perceptron itère sur les données un nombre prédéterminé d'époques E. Pour chaque exemple, l'analyseur compare la meilleure analyse qu'il prédit $\hat{\mathcal{C}}$, étant donnée la valeur courante du vecteur $\mathbf{w}$ de poids, avec l'analyse correcte $\mathcal{R}_i$. En cas d'erreur, l'algorithme met à jour les poids en pénalisant les poids associés à l'analyse prédite et en favorisant les poids associés à l'analyse correcte :

1: **function** PERCEPTRONLEARN($\mathbf{w}, (\mathcal{T}_1, \mathcal{R}_1) \ldots (\mathcal{T}_N, \mathcal{R}_N), E$)
2: $\quad \mathbf{w} \leftarrow 0^d$
3: $\quad \bar{\mathbf{w}} \leftarrow 0^d$
4: $\quad$ **for** $e = 1$ **to** E **do** ▷ Iterations sur E epoch
5: $\quad\quad$ **for** $i = 1$ **to** N **do** ▷ Iterations sur les données
6: $\quad\quad\quad \hat{\mathcal{C}} = \text{argmax}_{C_{0\Rightarrow 3n-1} \in \text{GEN}_{3n-1}(\mathcal{T}_i)}\ \mathbf{w} \cdot \mathbf{\Phi}_g(C_{0\Rightarrow 3n-1})$ ▷ Prédiction (analyse de le phrase)
7: $\quad\quad\quad$ **if** $\hat{\mathcal{C}} \neq \mathcal{R}_i$ **then** ▷ Mise à jour
8: $\quad\quad\quad\quad \mathbf{w} \leftarrow \mathbf{w} + \Phi_g(\mathcal{R}_i) - \Phi_g(\hat{\mathcal{C}})$
9: $\quad\quad\quad \bar{\mathbf{w}} \leftarrow \bar{\mathbf{w}} + \mathbf{w}$ ▷ Moyennage
10: $\quad$ **return** $\bar{\mathbf{w}}/(N \times E)$

L'algorithme du perceptron n'offre pas de garantie de convergence si les données ne sont pas linéairement séparables, ce qui est généralement le cas pour l'analyse syntaxique. Toutefois, en supposant que l'algorithme génère en cours d'analyse différents vecteurs de poids $\mathbf{w}^{(1)} \ldots \mathbf{w}^{(N \times E)}$ proches de la solution optimale du problème d'optimisation, nous utilisons un algorithme du perceptron moyenné dont l'estimation finale des poids est la moyenne de l'ensemble des vecteurs de poids générés en cours d'apprentissage (Collins, 2002).

Revenons sur l'usage du faisceau dans le cas de l'apprentissage structuré avec l'algorithme du perceptron. L'algorithme du perceptron fait l'hypothèse que chaque étape de prédiction renvoie l'analyse de poids le plus élevé pour un exemple donné (ligne 6), en utilisant l'équation (2). Or l'usage d'une méthode de prédiction approximative avec faisceau ne permet pas de garantir cette hypothèse. L'approximation par faisceau renvoie la solution à l'équation (4) et il n'est pas garanti que $\tilde{\mathcal{C}} = \hat{\mathcal{C}}$. Pire encore, il existe des cas où $\tilde{C} \neq \mathcal{R}$ alors que $\hat{C} = \mathcal{R}$: ce sont les cas où l'hypothèse $\hat{\mathcal{C}}$ n'est pas générée car écartée prématurément par le faisceau. Autrement dit, l'approximation introduite par le faisceau peut causer une mise à jour des poids invalide et par conséquent perturber significativement le processus d'estimation.

Pour contourner le problème, on utilise des méthodes de mise à jour sur des séquences d'analyse partielles à la suite de (Collins, 2002). Dans ce contexte (Huang *et al.*, 2012) démontre que, dans le cas structuré, pour garantir la convergence dans le cas linéairement séparable, la mise à jour du perceptron peut se réaliser à partir de sous-séquences d'analyse qui satisfont deux conditions affaiblies. Notons $C^{(r)}_{0\Rightarrow k}$ la dérivation de référence à l'étape k et $C^{(0)}_{0\Rightarrow k} = \text{argmax}_{C_{0\Rightarrow k} \in \text{GEN}^K_k(\mathcal{T})} W(C_{0\Rightarrow k})$ la meilleure sous séquence de dérivation dans le faisceau à l'étape k. Si $C^{(0)}_{0\Rightarrow k} \neq C^{(r)}_{0\Rightarrow k}$ et que $W(C^{(0)}_{0\Rightarrow k}) > W(C^{(r)}_{0\Rightarrow k})$ alors la mise à jour :

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{\Phi}_g(C^{(r)}_{0\Rightarrow k}) - \mathbf{\Phi}_g(C^{(0)}_{0\Rightarrow k}) \tag{7}$$

est valide. Garantir que la mise à jour est valide garantit la convergence de l'algorithme dans le cas où les données sont linéairement séparables. Dans cet article, nous examinons deux méthodes qui garantissent que la mise à jour est valide. Celles-ci diffèrent sur le choix effectif de k. La première méthode est l'*early update* (Collins, 2002) et dans ce cas, $k = \min\left(\{k | C^{(r)}_{0\Rightarrow k} \notin \text{GEN}^K_k(\mathcal{T})\}\right)$. La seconde est la *max violation update* (Huang *et al.*, 2012). En considérant l'ensemble des violations $V = \{k \mid C^{(0)}_{0\Rightarrow k} \neq C^{(r)}_{0\Rightarrow k}, W(C^{(0)}_{0\Rightarrow k}) > W(C^{(r)}_{0\Rightarrow k})\}$, on choisit $k = \text{argmax}_{k \in V}\ W(C^{(0)}_{0\Rightarrow k}) - W(C^{(r)}_{0\Rightarrow k})$

5 Le cas relâché

Dans la pratique, un certain nombre de cas d'utilisation de l'analyseur ne permettent pas d'utiliser facilement une grammaire 2-LCFG telle que supposée jusqu'ici. Il s'agit typiquement de cas où un étiqueteur morphosyntaxique est utilisé pour assigner des catégories aux mots. Dans ce type de cas, on peut souhaiter que l'analyseur utilise une séquence de tags $t_1 \dots t_n$ comme symboles terminaux. D'une part, les transformations de treebank que nous avons présentées en Section 2 peuvent potentiellement altérer ce jeu de tags (Figure 1). D'autre part, une procédure de transformation alternative qui garantit ne pas modifier le jeu de tags donne naturellement des arbres qui ne suivent pas strictement une forme normale de Chomsky (Figure 4).

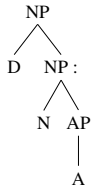


FIGURE 4 – Structure nouvelle dans le cas relâché

Certains terminaux sont introduits par des règles unaires, alors que d'autres sont introduits par des règles binaires. Par contraste avec 2-LCFG, ce type de configuration introduit naturellement de nouvelles règles de grammaire 2-CFG de la forme : $A \rightarrow B\,w, A \rightarrow w\,B$. Où w dénote un terminal pour cette grammaire (tag). Ces nouvelles formes de règles changent une propriété de 2-LCFG sur laquelle nous nous sommes appuyés jusqu'à présent : le nombre de pas de dérivation η pour dériver un arbre d'analyse d'une séquence de n mots est maintenant variable : $n-1 \leq \eta \leq 2n-1$. La conséquence est que les séquences de dérivation de l'analyseur ont une longueur η' telle que $2n-1 \leq \eta' \leq 3n-1$ et nous observons en pratique que le modèle a un biais naturel pour les séquences plus longues : celles-ci ont généralement un poids plus élevé. Pour traiter ce biais potentiel, nous formulons deux variantes de l'analyseur. La première variante, "naïve", consiste simplement à modifier la condition de terminaison de l'analyseur. Pour cela on redéfinit l'ensemble des configurations terminées $Succ = \{C_{0\Rightarrow k} | C_k = \langle n+1, \langle \sigma_e, \tau \rangle \rangle, 2n-1 \leq k \leq 3n-1\}$. Dans ce contexte, l'Equation (4) se reformule comme suit :

$$\tilde{\mathcal{C}} = \underset{C_{0\Rightarrow k} \in Succ}{\operatorname{argmax}} \; W(C_{0\Rightarrow k}) \tag{8}$$

La seconde variante de l'algorithme, dite "synchronisée", a pour but de garantir que $\eta' = 3n-1$ dans le cas pratique traité ici. Pour ce faire nous contraignons l'algorithme à réaliser nécessairement une réduction unaire ou une réduction fantôme après avoir décalé un terminal. Ce type de contrainte s'exprime en modifiant légèrement la méthode de compilation de l'automate LR décrite en Section 3. Une réduction fantôme est une nouvelle action de l'analyseur et la règle d'inférence, notée GR, associée à cette action est donnée en Figure 3. Cette règle est conçue pour synchroniser la procédure d'inférence sans changer significativement le contenu de la pile. La règle fait en quelque sorte perdre un temps à l'analyseur dans le cas où il choisit de ne pas faire de réduction unaire. Cette seconde variante ne demande pas de modifier l'équation (4).

6 Représentation structurée des mots et spécification d'un modèle d'analyse

La dernière extension que nous introduisons est motivée par la volonté de modéliser des langues à morphologie plus riche que l'anglais ou le chinois, comme par exemple le français. Pour cette famille de langues, les mots sont non seulement caractérisés par une forme mais également par un vecteur de propriétés morphologiques comme des marques d'accord en genre et en nombre, de temps, de mode, de personne, voire de cas. La richesse morphologique de ces langues contribue naturellement à augmenter le nombre de formes de mots, ce qui accroit les phénomènes de dispersion statistique. Pour cette raison, on fait l'hypothèse que l'usage d'informations extraites de dictionnaires (Mirroshandel *et al.*, 2013), de formes lemmatisées (Seddah *et al.*, 2010) ou d'aggrégats distributionnels (Candito & Crabbé, 2009) pour compléter les données d'un treebank constituent une source d'information utile à un modèle d'analyse.

Le modèle d'analyse présenté jusqu'à présent manipule les mots comme des objets atomiques encodés dans les catégories lexicalisées de la grammaire 2-LCFG. L'extension décrite ici consiste à autoriser le codage des mots par des tuples structurés de taille arbitraire. En notant $\boldsymbol{\omega}$ un tel tuple, les catégories de la grammaire 2-LCFG sont maintenant codées par des symboles de la forme $A[\boldsymbol{\omega}]$. Dans cette version étendue, les traits peuvent ainsi accéder aux différents champs de ces tuples lors de l'exécution. La taille de ces tuples est laissée libre à l'utilisateur et est en pratique fonction de la richesse des données dont il dispose en entrée du processus d'analyse. On donne à titre d'exemple indicatif en Figure 5 une représentation possible en 2-LCFG

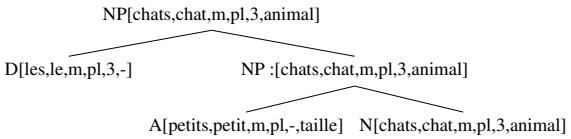


FIGURE 5 – Propagation guidée par les têtes de structures de traits lexicales

$s_0t.w_c \,\&\, s_0t.c$	$s_0t.w_f \,\&\, s_1t.w_f$	$s_0t.c \,\&\, s_1t.c \,\&\, s_2t.c$	$s_0t.c \,\&\, q_2.w_c \,\&\, q_3.w_c$	**Accord**
$s_0t.w_f \,\&\, s_0t.c$	$s_0t.w_f \,\&\, s_1t.c$	$s_0t.w_f \,\&\, s_1t.c \,\&\, s_2t.c$	$s_0t.c \,\&\, q_2.w_f \,\&\, q_3.w_c$	$s_0tc \,\&\, e(s_0t.agr, s_1t.agr) \,\&\, s_1t.c$
$s_1t.w_c \,\&\, s_1t.c$	$s_0t.c \,\&\, s_1t.w_f$	$s_0t.c \,\&\, s_1t.w_f \,\&\, q_0.w_c$	$s_0t.c \,\&\, q_2.w_c \,\&\, q_3.w_f$	$s_0tc \,\&\, e(s_0t.num, s_1t.num) \,\&\, s_1t.c$
$s_1t.w_f \,\&\, s_1t.c$	$s_0t.c \,\&\, s_1t.c$	$s_0t.c \,\&\, s_1t.c \,\&\, s_2t.w_f$	$s_0t.c \,\&\, s_0r.c \,\&\, s_1t.c$	$s_0tc \,\&\, e(s_0t.gen, s_1t.gen) \,\&\, s_1t.c$
$s_2t.w_c \,\&\, s_2t.c$	$s_0t.w_f \,\&\, q_0.w_f$	$s_0t.c \,\&\, s_1t.c \,\&\, q_0.w_c$	$s_0t.c \,\&\, s_0r.c \,\&\, s_1t.w_f$	$s_0tc \,\&\, e(s_0t.agr, q_0.agr) \,\&\, q_1.w_c$
$s_2t.w_c \,\&\, s_2t.c$	$s_0t.c \,\&\, q_0.w_f$	$s_0t.w_f \,\&\, s_1t.c \,\&\, q_0.w_c$	$s_0t.w \,\&\, s_0r.c \,\&\, s_1t.w_f$	$s_0tc \,\&\, e(s_0t.gen, q_0.gen) \,\&\, q_1.w_c$
$q_0.w_c \,\&\, q_0.w_f$	$s_0t.c \,\&\, q_0.w_c$	$s_0t.c \,\&\, s_1t.w_f \,\&\, q_0.w_c$	$s_0t.c \,\&\, s_0l.w_f \,\&\, s_1t.c$	$s_0tc \,\&\, e(s_0t.num, q_0.num) \,\&\, q_1.w_c$
$q_1.w_c \,\&\, q_1.w_f$	$q_0.w_f \,\&\, q_1.w_f$	$s_0t.c \,\&\, s_1t.c \,\&\, q_0.w_f$	$s_0t.c \,\&\, s_0l.c \,\&\, s_1t.w_f$	$s_0tc \,\&\, e(s_0t.agr, q_1.agr) \,\&\, q_1.w_c$
$q_2.w_c \,\&\, q_2.w_f$	$q_0.w_f \,\&\, q_1.w_c$	$s_0t.c \,\&\, q_0.w_c \,\&\, q_1.w_c$	$s_0t.c \,\&\, s_0l.c \,\&\, s_1t.c$	$s_0tc \,\&\, e(s_0t.num, q_0.num) \,\&\, q_1.w_c$
$q_3.w_c \,\&\, q_3.w_f$	$q_0.w_c \,\&\, q_1.w_c$	$s_0t.c \,\&\, q_0.w_f \,\&\, q_1.w_c$	**Mode**	$s_0tc \,\&\, e(s_0t.gen, q_0.gen) \,\&\, q_1.w_c$
$s_0l.w_f \,\&\, s_0l.c$	$s_1t.w_f \,\&\, q_0.w_f$	$s_0t.c \,\&\, q_0.w_c \,\&\, q_1.w_f$	$s_0t.w_m \,\&\, s_1t.w_f$	**Sous − cat**
$s_0r.w_f \,\&\, s_0r.c$	$s_1t.w_f \,\&\, q_0.w_c$	$s_0t.c \,\&\, q_1.w_c \,\&\, q_2.w_c$	$s_0t.w_f \,\&\, s_1t.w_m$	$s_0t.w_X \,\&\, s_1t.w_f$
$s_1l.w_f \,\&\, s_1l.c$	$s_1t.c \,\&\, q_0.w_f$	$s_0t.c \,\&\, q_1.w_f \,\&\, q_2.w_c$	$s_0t.c \,\&\, s_1t.w_m$	$s_0t.w_f \,\&\, s_1t.w_X$
$s_1r.w_f \,\&\, s_1r.c$	$s_1t.c \,\&\, q_0.w_c$	$s_0t.c \,\&\, q_1.w_c \,\&\, q_2.w_f$	$s_0t.w_m \,\&\, s_1t.c$	$s_0t.c \,\&\, s_1t.w_X$ $\quad s_0tw_X \,\&\, s_1t.c$

FIGURE 6 – Spécification des modèles d'analyse

relâchée d'un groupe nominal pour le français où les mots sont encodés par des tuples.

Comme il est d'usage dans la plupart des modèles de TAL, des gabarits (*templates*) sont utilisés comme langage destiné à spécifier l'ensemble des fonctions features ϕ_i à valeurs booléennes du modèle d'analyse. Nous utilisons les notations suivantes pour spécifier les gabarits du modèle d'analyse explicités en Figure (6). Les gabarits permettent d'adresser des valeurs observées relativement à une configuration de l'analyseur. Avant le point on note l'adressage d'un noeud dans une configuration. Ainsi $s_it, s_il, s_ir (0 \leq i \leq 2)$ dénotent les noeuds de la pile d'exécution avec s_0 le sommet de la pile. Un noeud de la pile s_i encode un arbre local de profondeur 1 (Figure 2) dont les différents noeuds sont adressés par les symboles t (racine), l (noeud de gauche), r (noeud de droite). q_i dénote les éléments de la file d'attente, avec q_0 le premier élément de cette file. Après le point, on note quelle valeur on extrait du noeud. c dénote la catégorie du constituant, w_f et w_c le mot tête et la catégorie du mot tête de ce constituant. On note de plus w_m, w_X le mode du mot tête et la sous-catégorie de ce mot (au sens du French Treebank), num,gen, le nombre et le genre de la tête, et agr les traits conjoints de genre et de nombre. Les gabarits élémentaires sont conjoints par le symbole $\&$ et finalement la notation $e(\cdot, \cdot)$ dénote une fonction qui renvoie vrai si les valeurs passées en argument sont égales.

7 Expériences

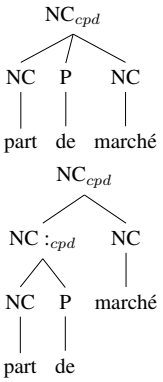
Nous présentons ici quelques expériences qui cherchent à mettrer en évidence le rôle des différents modules de l'analyseur. Celles-ci permettent également de comparer l'analyseur à l'état de l'art.

Protocole Les expériences s'appuient sur le jeux de données français SPMRL décrit dans (Abeillé *et al.*, 2003; Seddah *et al.*, 2013). Celles-ci devraient constituer le nouveau jeu de données standard pour l'analyse en constituants du français dans les années à venir et représentent un cadre plus réaliste que ceux utilisés précédemment par (Crabbé & Candito, 2008) dans la mesure où il faut également analyser les mots composés. Le jeu de données SPMRL instancie les données French Treebank dans deux scénarios : le scénario 'tags prédits' comporte un jeu de test où les tags de référence sont remplacés par des tags prédits par un tagger (exactitude d'étiquetage = 97.35%) et un scénario 'tags donnés' où le jeu de test comporte les tags de référence.

Les expériences sont menées avec une implantation de l'algorithme décrit dans l'article, écrite en C++. En particulier nous utilisons systématiquement un noyau de hachage et les mémo-fonctions décrites ci-dessus. Les données sont binarisées par une markovisation par la tête d'ordre 0, les têtes sont assignées par les heuristiques de (Arun & Keller, 2005), nous avons de plus assigné comme tête à une structure de mot composé son fils le plus à gauche. Les gabarits utilisés par défaut sont ceux spécifiés en Figure 6. Les expériences sont réalisées sur les données de développement et la comparaison avec l'analyseur de (Petrov *et al.*, 2006) est réalisée sur les données de test. Nous mesurons le F-Score et la couverture sur les données débinarisées à l'aide du logiciel `evalb` [2] et les temps reportés sont mesurés sur le jeu de test. Ils sont mesurés en secondes par phrase sur une même machine (MacOSX 2.4Ghz) pour chacun des analyseurs sans tenir compte du temps de lecture et d'écriture des données. Nous mesurons le F-Score sur les mots composés $F(\text{cpd})$ à l'aide de l'évaluateur intégré à l'analyseur de Stanford.

Chaque expérience menée fait varier une seule variable expérimentale par contraste avec une configuration de l'analyseur

2. Nous utilisons la version standard du logiciel telle que distribuée sur le site `http://nlp.cs.nyu.edu/evalb`.



Règles de dominance immédiate pour G_m^{base}		
$[a] \rightarrow [n]\,[n]$	$[n] \rightarrow [n]\,,\,[n]$	$t \rightarrow [n]\,,\,[n] \quad (\forall t \in T)$
$[a] \rightarrow [n]\,,\,[w]$	$[n] \rightarrow [n]\,,\,[w]$	$t \rightarrow [n]\,,\,[w] \quad (\forall t \in T)$
$[a] \rightarrow [w]\,[w]$	$[n] \rightarrow [w]\,,\,[w]$	$t \rightarrow [w]\,,\,[w] \quad (\forall t \in T)$
$[a] \rightarrow [w]$	$[n] \rightarrow [w]$	$t \rightarrow [n]\,,\,t \quad (\forall t \in T)$
$[a] \rightarrow [n]\,,\,t \quad (\forall t \in T)$	$[n] \rightarrow [n]\,,\,t \quad (\forall t \in T)$	$t \rightarrow [w]\,,\,t \quad (\forall t \in T)$
$[a] \rightarrow [w]\,,\,t \quad (\forall t \in T)$	$[n] \rightarrow [w]\,,\,t \quad (\forall t \in T)$	
Règles de dominance immédiate supplémentaires pour G_m^{cpd}		
$[a] \rightarrow [n]_{cpd}\,,\,[n]$	$[n] \rightarrow [n]_{cpd}\,,\,[n]$	$t \rightarrow [n]_{cpd}\,,\,[n] \quad (\forall t \in T)$
$[a] \rightarrow [n]_{cpd}\,,\,[n]_{cpd}$	$[n] \rightarrow [n]_{cpd}\,,\,[n]_{cpd}$	$t \rightarrow [n]_{cpd}\,,\,[n]_{cpd} \quad (\forall t \in T)$
$[a] \rightarrow [n]_{cpd}\,,\,[w]$	$[n] \rightarrow [n]_{cpd}\,,\,[w]$	$t \rightarrow [n]_{cpd}\,,\,[w] \quad (\forall t \in T)$
$[a] \rightarrow [n]_{cpd}\,,\,t \quad (\forall t \in T)$	$[n] \rightarrow [n]_{cpd}\,,\,t \quad (\forall t \in T)$	$t \rightarrow [n]_{cpd}\,,\,t \quad (\forall t \in T)$
$[n]_{cpd} \rightarrow [w]\,,\,[w]$	$[n]_{cpd} \rightarrow t_{cpd}\,,\,[w] \quad (\forall t_{cpd} \in T_{cpd})$	$t_{cpd} \rightarrow t_{cpd}\,,\,[w] \quad (\forall t_{cpd} \in T_{cpd})$
$[n]_{cpd} \rightarrow [w]$	$t_{cpd} \rightarrow [w]\,,\,[w] \quad (\forall t_{cpd} \in T_{cpd})$	$t_{cpd} \rightarrow [w] \quad (\forall t_{cpd} \in T_{cpd})$

FIGURE 7 – Représentation structurée des mots composés (SPMRL) et grammaires matrices correspondantes. *En haut à gauche, on a la représentation des mots composés dans le jeu de données* SPMRL. *La structure binarisée correspondante est donnée en bas à gauche. La grammaire générale $G_m^{(base)}$ encode une grammaire matrice qui ne tient pas spécialement compte des mots composés et dont les classes d'équivalence sont $[a]$ le symbole axiome, $[n]$ les symboles non terminaux non temporaires et $[w]$ l'ensemble des terminaux. Chaque non terminal temporaire $t \in T$ forme sa propre classe d'équivalence. La grammaire $G_m^{(cpd)}$ distingue en plus une classe d'équivalence $[n]_{cpd}$, représentant les non terminaux marqués comme composés, disjointe de la classe $[n]$ des terminaux non marqués comme tels. $T_{(cpd)}$ est un ensemble de symboles temporaires marqués comme composés qui est disjoint de T.*

par défaut. La configuration par défaut pose que la taille du faisceau $K = 4$, que la gestion du relâchement utilise la version naïve (Section 5), que la grammaire utilise la construction LR $G_m^{(base)}$ (Section 3 et Figure 7), que la mise à jour est l'*early update* (Section 4) et que l'intégralité des gabarits donnée en Figure 6 est utilisée. Le protocole spécifique à chaque expérience est le suivant. *(1) Taille du Beam* La première expérience fait varier la taille du faisceau. Nous testons différentes valeurs de la constante K qui fixe la taille du faisceau ($\text{GEN}_k^K(\mathcal{T})$ ci-dessus) pour $K = 2, K = 4$ (défaut), $K = 8, K = 16$. *(2) Relâchement* Par défaut, nous utilisons l'analyseur en mode naïf comme spécifié en Section 5 dans les différentes expériences. Pour l'expérience de relâchement, nous testons le mode naïf (*naive*) et le mode synchronisé (*sync*) comme décrit en Section 5. *(3) Grammaire* Nous faisons varier le type de grammaire matrice utilisée (Section 3) pour générer l'automate LR en utilisant les grammaires matrices présentées en (Figure 7). Nous contrastons en particulier une grammaire générale $G_m^{(base)}$ avec une grammaire $G_m^{(cpd)}$ qui comporte une sous-grammaire spécifique pour le traitement des mots composés. *(4) Mise à jour* Cette expérience fait varier la méthode de mise à jour comme décrit en Section 4. On teste l'influence de la mise à jour rapide (*early update*) par contraste avec la mise à jour à violation maximale (*max violation*). Les modèles entrainés avec la mise à jour rapide le sont sur 25 époques. Les modèles avec mise à jour à violation maximale sont entraînés sur 12 époques. *(5) Morphologie* Cette expérience teste l'impact des gabarits morphologiques. On contraste le modèle qui comporte l'ensemble des gabarits morphologiques (Figure 6), modélisant notamment l'accord avec un modèle plus pauvre ou les gabarits rangés sous les sections *accord, sous-cat, mode* en Figure 6 sont ignorés. *(6) Analyseur de Berkeley* Il s'agit de comparer l'analyseur décrit dans cet article avec l'analyseur de Berkeley (Petrov *et al.*, 2006) connu pour représenter l'état de l'art en termes de rapidité et de correction des analyses. Nous réutilisons ici les résultats donnés par (Seddah *et al.*, 2013) sur le jeu de test avec cet analyseur en utilisant le score `evalb` standard.

Résultats et discussions Très généralement, les expériences contribuent à justifier la méthode d'analyse proposée dans cet article. Celle-ci se fonde sur un algorithme à décalage réduction et une méthode d'inférence approximative en faisceau pour l'analyse en constituants. Les résultats obtenus sont état de l'art en temps comme en exactitude, en excluant divers résultats obtenus par mélange d'analyseurs et par utilisation de ressources exogènes. Si en ce qui concerne l'exactitude, les différences avec l'analyseur de Berkeley (Petrov *et al.*, 2006) sont faibles (Table 3), on constate que l'apport principal de la méthode décrite ici est son efficacité en temps (Figure 8).

On constate plus spécifiquement que c'est la prise en compte des informations morphologiques qui permet à l'analyseur d'être état de l'art $F_{\leq 40} = 87.02$ pour $K = 4$ et $F_{\leq 40} = 87.14$ pour $K = 8$ avec des faisceaux de petite taille et des temps d'analyse très faibles ($t_\mu = 0.06s, t_{\text{max}} = 0.3s$ pour $K = 4$. et $t_\mu = 0.1s, t_{\text{max}} = 0.5s$ pour $K = 8$) par comparaison avec l'existant : Berkeley (Petrov *et al.*, 2006) $t_\mu = 0.28s, t_{\text{max}} = 10.27s$. Ce dernier est reporté par (Huang & Sagae, 2010) comme plus rapide que (Charniak, 2000). Autrement dit l'expressivité ajoutée par le modèle permet de compenser l'approximation introduite pour le rendre efficace.

Dev(tags donnés)			
Expérience	F≤ 40	F	Cov
K=2	85.40	82.74	98.6
K=4	86.52	83.69	99.5
K=8	86.80	84.31	99.9
K=16	86.49	83.95	99.9
no-morph	85.23	82.43	99.8
all-morph	86.52	83.69	99.5
sync	86.41	83.66	99.6
naive	86.52	83.69	99.5
cpd	86.30	83.30	99.2
base	86.52	83.69	99.5
Max Violation	85.98	83.49	99.5
Early Update	86.52	83.69	99.5

Dev(tags prédits)			
Expérience	F≤ 40	F	Cov
K=2	83.24	80.42	98.9
K=4	84.32	81.34	99.4
K=8	84.43	81.79	99.8
K=16	84.59	81.94	99.8
no-morph	83.68	81.05	99.8
all-morph	84.32	81.34	99.4
sync	84.06	81.14	99.9
naive	84.32	81.34	99.4
cpd	83.84	81.17	99.0
base	84.32	81.34	99.4
Max Violation	83.42	80.56	99.5
Early Update	84.32	81.34	99.4

Test (tags donnés)	F≤ 40	F	Cov	F(cpd)
K=4	87.02	83.99	99.7	77.48
K=8	87.14	84.20	99.8	77.26
Berkeley	86.44	83.96	99.9	73.38
Test (tags prédits)	F≤ 40	F	Cov	F(cpd)
K=4	84.03	80.98	99.7	69.39
K=8	84.33	81.43	99.8	70.26
Berkeley	83.16	80.73	99.9	69.32
Test (texte brut)	F≤ 40	F	Cov	F(cpd)
Berkeley	83.59	81.33	99.9	70.25

TABLE 3 – Résultats expérimentaux

Il est par contre plus surprenant de constater le résultat nul concernant la synchronisation de l'analyseur. La version naïve se comporte même un peu mieux que la version explicitement synchronisée. Il est possible que le jeu de gabarits qui a été mis au point principalement sur le modèle d'exécution naïf procure un avantage à ce dernier.

On remarque que le modèle comportant une grammaire locale spécifique aux mots composés $G_m^{(cpd)}$ a une couverture plus faible que la grammaire $G_m^{(base)}$. Par contre on remarque qu'il est significativement plus rapide ($t_\mu = 0.04s, t_{\max} = 0.2s$) . La grammaire est plus contrainte et l'automate comporte moins d'états de succès. On pense travailler à l'avenir pour définir une méthode de recherche de solutions garantissant la complétude de l'algorithme d'analyse pour permettre de combiner la grammaire générale avec des grammaires locales pour mots composés (ou entités nommées par exemple).

En ce qui concerne, la méthode de mise à jour, l'observation principale concerne la courbe d'apprentissage. L'algorithme converge beaucoup plus vite pour la méthode *Max Violation* (entre 10 et 15 itérations contre 25-30 pour *Early update*). Par contre la méthode *Max Violation* a une tendance au surentrainement encore plus prononcée que *Early Update*, ce qu'il faut améliorer à l'avenir.

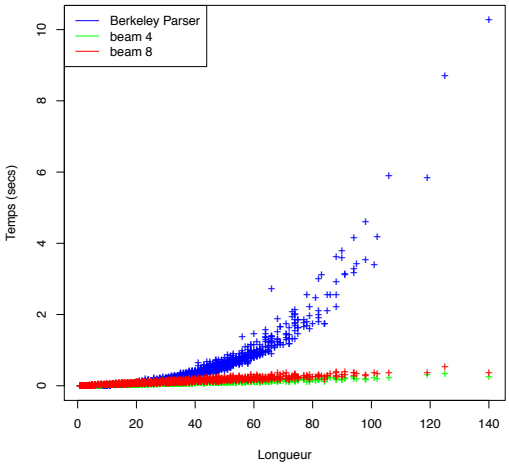


FIGURE 8 – Temps d'analyse

8 Conclusion

À notre connaissance, l'article propose le premier algorithme d'analyse syntaxique déterministe en constituants pour le langage naturel guidé par un automate LR et pondéré par un modèle discriminant. L'article montre que l'usage d'un modèle d'analyse très expressif, qui permet notamment de capturer des informations morphologiques, est rendu très efficace par l'usage d'une approximation heuristique. Celle-ci permet d'obtenir une exécution en temps linéaire tout en obtenant des résultats état de l'art en exactitude pour l'analyse syntaxique du français. La suite des travaux va porter principalement sur l'intégration d'une construction sémantique conjointe à l'analyse syntaxique et sur l'intégration de ressources exogènes dans le modèle d'analyse dans le but de créer un analyseur sémantique capable d'analyser efficacement de gros volumes de données textuelles issues du web.

Remerciements

L'auteur remercie Maximin Coavoux qui a contribué au développement, Benoît Sagot pour ses encouragements et les diverses discussions ayant permis de clarifier le propos, et finalement Djamé Seddah pour son aide sur les données.

Références

ABEILLÉ A., CLÉMENT L. & TOUSSENEL F. (2003). Building a treebank for French. In *Treebanks*. Kluwer.

AHO A. V., SETHI R., ULLMAN J. D. & LAM M. S. (2006). *Compilers : Principles, Techniques, and Tools*. Addison Wesley.

ARUN A. & KELLER F. (2005). Lexicalization in crosslinguistic probabilistic parsing : The case of French. In *Association for Computational Linguistics*.

CANDITO M. & CRABBÉ B. (2009). Improving generative statistical parsing with semi-supervised word clustering. In *Proceedings of International Workshop on Parsing Technologies (IWPT)*.

CHARNIAK E. (2000). A maximum-entropy-inspired parser. In *North American Association for Computational Linguistics*.

CHARNIAK E. & JOHNSON M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*.

COLLINS M. (2002). Discriminative training methods for hidden markov models : Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

COLLINS M. (2003). Head-driven statistical models for natural language parsing. *Computational Linguistics*, **29**(4).

CRABBÉ B. & CANDITO M. (2008). Expériences d'analyses syntaxique statistique du français. In *Actes deTALN 2008*.

FINKEL J. R., KLEEMAN A. & MANNING C. D. (2008). Efficient, feature-based, conditional random field parsing. In *Proceedings of the Association for Computational Linguistics*.

GAZDAR G., KLEIN E., PULLUM G. K. & SAG I. A. (1985). *Generalized Phrase Structure Grammar*. Cambridge, MA : Harvard University Press and Oxford : Basil Blackwell's.

HUANG L., FAYONG S. & GUO Y. (2012). Structured perceptron with inexact search. In *North American Association for Computational Linguistics*.

HUANG L. & SAGAE K. (2010). Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

KNUTH D. (1965). On the translation of languages from left to right. *Information and Control*, **8**(6).

MIRROSHANDEL S. A., NASR A. & SAGOT B. (2013). Enforcing subcategorization constraints in a parser using subparses recombining. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

NEDERHOF M.-J. & SATTA G. (2010). Algorithmic aspects of natural language processing. In M. ATALLAH & M. BLANTON, Eds., *Algorithms and Theory of Computation Handbook*. CRC press.

PETROV S., BARRETT L., THIBAUX R. & KLEIN D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia : Association for Computational Linguistics.

SAGAE K. & LAVIE A. (2006). A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL*.

SEDDAH D., CHRUPALA G., CETINOGLU O., VAN GENABITH J. & CANDITO M. (2010). Lemmatization and lexicalized statistical parsing of morphologically rich languages : the case of French. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically Rich Languages*.

SEDDAH D., TSARFATY R., KÜBLER S., CANDITO M., CHOI J. D., FARKAS R., FOSTER J., GOENAGA I., GOJENOLA GALLETEBEITIA K., GOLDBERG Y., GREEN S., HABASH N., KUHLMANN M., MAIER W., NIVRE J., PRZEPIÓRKOWSKI A., ROTH R., SEEKER W., VERSLEY Y., VINCZE V., WOLIŃSK M., WRÓBLEWSKA A. & VILLEMONTE DE LA CLERGERIE É. (2013). Overview of the SPMRL 2013 Shared Task : A Cross-Framework Evaluation of Parsing Morphologically Rich Languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*.

SHI Q., DROR G., LANGFORD J., SMOLA A. & VISHWANATHAN S. (2009). Hash kernels for structured data. *Journal of Machine Learning Research*, **10**.

SOCHER R., HUVAL B., MANNING C. D. & NG A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces. In *Conference on Empirical Methods in Natural Language Processing*.

TOMITA M. (1985). An efficient context free parsing algorithm for natural language. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

TOMITA M. (1988). Graph structured stack and natural language parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*.

ZHANG Y. (2009). Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings International Workshop on Parsing Technologies*.

ZHU M., ZHANG Y., CHEN W., ZHANG M. & ZHU J. (2013). Fast and accurate shift-reduce constituent parsing. In *Association for Computational Linguistics*.