An LR-inspired generalized lexicalized phrase structure parser

Benoit Crabbé
ALPAGE / INRIA - Université Paris Diderot
Place Paul Ricoeur
F-75013 Paris
benoit.crabbe@univ-paris-diderot.fr

Abstract

The paper introduces an LR-based algorithm for efficient phrase structure parsing of morphologically rich languages. The algorithm generalizes lexicalized parsing (Collins, 2003) by allowing a structured representation of the lexical items. Together with a discriminative weighting component (Collins, 2002), we show that this representation allows us to achieve state of the art accurracy results on a morphologically rich language such as French while achieving more efficient parsing times than the state of the art parsers on the French data set. A comparison with English, a lexically poor language, is also provided.

1 Introduction

The paper provides a phrase structure parsing algorithm inspired by LR (Knuth, 1965), GLR (Tomita, 1988) and the recent developments of (Huang and Sagae, 2010) for dependency grammar. The parsing algorithm comes with a discriminative weighting framework inspired by (Collins, 2002). Although discriminative phrase structure parsing has been shown to be challenging when it comes to efficiency issues (Turian and Melamed, 2006; Finkel et al., 2008), we use here several approximations that make the framework not only tractable but also efficient and accurate on a lexically rich language such as French.

Despite the successes of dependency grammar, we are interested in phrase structure grammar since it naturally allows to support compositional semantic representations as recently highlighted by (Socher et al., 2012). It remains that most phrase structure parsers have been designed in priority for modelling lexically poor languages such as English or Chinese (Collins, 2003; Charniak, 2000; Zhu et al., 2013). Although highly accurate multilingual parsers exist (Petrov et al., 2006), they remain relatively both slow for wide coverage purposes and their inner formal structure is not designed to handle naturally morphological information.

We assume that parsing lexically rich languages benefits from taking into account the structured morphological information that can be extracted from lexical forms. Using French as a case study we show that we can reach both parsing efficiency with an approximative inference method and we can get a state of the art accurracy by generalizing lexicalized parsing to handle feature structure-based word representations. Our proposal also differs theoretically from related ones (Sagae and Lavie, 2006; Zhang and Clark, 2011; Zhu et al., 2013) by explicitly using an LR automaton. The explicit introduction of the LR automaton allows us to establish a formal difference between shift reduce phrase structure parsing and shift reduce dependency parsing. It further provides some insights on the nature of the grammar underlying many contemporary parsers.

The paper is organized as follows. First, section 2, we set up a formal framework for describing weighted phrase structure parsing as a 2-LCFG (Nederhof and Satta, 2010). Observing that the tree structures are actually constrained in practice we formulate in section 3 an LR automaton construction method for treebank grammars suitable for encoding these constraints. We then provide in section 4 a description of the algorithm and its components. Section 5 give an extension to 2-LCFG suitable for parsing morphologically rich languages and meeting common practical requirements. The whole

Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 541–552, Dublin, Ireland, August 23-29 2014.

framework is then evaluated in section 6 on French and English allowing to better identify its properties with respects to the state of the art.

2 Grammatical representation

The first step we consider is the grammar actually used for parsing and how it is generated. We suppose here a bilexical context free grammar or 2-LCFG (Nederhof and Satta, 2010). A 2-LCFG is a CFG whose rules are of the form given in Figure 1 (left). Symbols of the form x and h denote terminal symbols while symbols of the form $A[h]$ or $A[x]$ denote lexicalized non terminals. A, B, C are non lexicalized non terminals and h denotes a head. A 2-LCFG rule is typically of the form $NP[cat] \rightarrow D[a]\, N[cat]$.

For practical robust parsing, 2-LCFG are grammars with a very large number of rules generated dynamically at runtime (Section 4). Most of the static grammatical preprocessing involved for the generation of an LR automaton only applies to the underlying delexicalized 2-CFG by ignoring lexical annotation symbols.

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

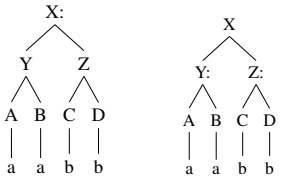


Figure 1: 2-LCFG rule patterns and invalid 2-CFG trees

The first step towards robust parsing thus requires to generate a grammar suitable for this purpose. In our case, the grammar is a treebank grammar and since most treebanks do encode trees with variable branching arities we must transform it to match the 2-LCFG required pattern. The first step amounts to apply an order 0 head markovization (Collins, 2003) which is followed by a reduction of unary rules. Both transformations guarantee that the trees do follow strictly a Chomsky Normal Form (CNF). Trees in CNF have two properties of interest. First, one can show by induction that they can be generated with a constant number of derivation steps η for a sentence of length $n : \eta = 2n-1$. This property is in principle critical for the comparison of weighted parsing hypotheses (Section 5) and explains why we use 2-LCFG as a grammatical representation in the first place. Second, the binarization (markovization) procedure also introduces temporary symbols we consider to be different from other non terminal symbols. These temporary symbols are further constraining the tree structure. Using ':' to denote a temporary symbol in Figure 1 (right), we observe for instance that the root of a tree cannot be temporary and two siblings cannot be temporaries either. By contrast, arc standard dependency parsers such as the one of (Huang and Sagae, 2010) do verify the first property while the second property is irrelevant in that case.

3 LR automaton construction

We use an LR automaton to enforce the parser to generate parse trees satisfying the above mentioned structural constraints. Although, other proposals such as (Sagae and Lavie, 2006) apparently returns a failure when the parser generates invalid trees and (Zhu et al., 2013) apparently handles the problem with local constraints preventing the parser to generate invalid configurations, we use here an $LR(0)$ automaton to ensure that the parser globally enforces these constraints. This seemed to us theoretically justified, easier to generalize (Section 6) and easier to implement.

As such, a traditional $LR(0)$ parser (Knuth, 1965) is not suited for parsing natural language: it aims to statically eliminate ambiguity from the grammar. Here, following (Tomita, 1985) the LR tables are built without trying to resolve conflicts. Instead the conflicts are kept and determinism is brought by a weighting component. The use of $LR(0)$ tables aims to ensure that the parser actually generates valid parse derivations. In this case, the generation of the derivations requires to constrain the underlying 2-CFG grammar with respects to temporary symbols. This being said, building an $LR(0)$ automaton for robust treebank grammars raise two issues. The first is inductive, a grammar read off from a treebank is not guaranteed to be robust and to generalize to other text, since a treebank remains a finite sample of language. The second is practical : traditional $LR(0)$ compilation methods involve the determinisation of

the LR NFA which is exponential in the number n of states of this NFA. In case of very large ambiguous treebank grammars n is very large and the compilation becomes intractable (Briscoe and Carroll, 1993).

These two observations lead us to design this automata by the following construction. First, let Σ be the set of non terminal symbols read off from the treebank, T be the set of temporaries introduced by binarization and N the set of non temporary symbols such that $\Sigma = N \cup T$ and $N \cap T = \emptyset$. Second we note W the set of terminal symbols extracted from the treebank and $A \in N$ the unique axiom of this grammar. We then partition Σ with the following set of equivalence classes: $[a] = \{A\}, [t] = T$ and $[n] = \Sigma - (T \cup A)$. For convenience we also note $[w] = W$. Given these equivalence classes, we define the matrix grammar $G_m = \langle \Sigma_m, [w], [a], R_m \rangle$ (where $\Sigma_m = \{[a], [n], [t]\}$). The rules R_m of G_m are then designed to enforce the above mentioned tree well formedness constraints. Some possible such rules are given in Table 1 using ID/LP notation (Gazdar et al., 1985).In other words, an immediate dominance rule of the form $a \rightarrow b\,,\, c$ is expanded as two rules $a \rightarrow b\,c$ and $a \rightarrow c\,b$. Such a grammar allows

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

Table 1: Example of Immediate Dominance rules for G_m

to enforce the above-mentioned constraints, it is also small and it is robust : $L(G_m) = [w]+$. We can then very easily build a deterministic $LR(0)$ automaton $A_m = \langle \Sigma_m \cup \{[w]\}, Q, i, F, E_m \rangle$ with classical methods (Aho et al., 2006). From this automaton we can then efficiently generate an expanded automaton $A_{exp} = \langle \{\Sigma \cup W\}, Q, i, F, E \rangle$ where $E = \{(q, a, q') \mid (q, [x], q') \in E_m, \forall a \in [x]\}$. In order to read off the $LR(0)$ table from A_{exp}, we consider the set of actions $\mathcal{A} \stackrel{def}{=} \{RL(X) | X \in \Sigma\} \cup \{RR(X) | X \in \Sigma\} \cup \{RU(X) | X \in \Sigma\} \cup \{S\}$ first introduced by (Sagae and Lavie, 2006). In short S denotes the shift action, $RU(X)$ denotes an unary reduction by terminal X, $RL(X)$ denotes a binary reduction by terminal X with left symbol marked as head, and $RR(X)$ denotes a binary reduction by terminal X with right symbol marked as head. By contrast with a classical LR action set, we extract the actions $RL(X)$ and $RR(X)$ from a state $q \in Q$ if we have an LR item of the form $\langle X \rightarrow A\,B\bullet \rangle$ without requiring that $\langle X \rightarrow B\,A\bullet \rangle \in q$. This simplification, mirrorring that of (Sagae and Lavie, 2006), reduces the number of actions, eases learning and makes parsing more efficient. This being said, the matrix grammar G_m given in Table 1 is not the only one possible (see also section 6). A valid rule set must enforce tree well formedness constraints by building upon a partition of Σ in equivalence classes. On the other hand the action set $\mathcal{A}$ defined here implies that for every rule $R \in R_m$ of the form $[x] \rightarrow [y]\,[z]$ there is a rule $R' \in R_m$ of the form $[x] \rightarrow [z]\,[y]$. That is why we formulate the rules R_m with ID/LP notation and this also means that we cannot express any word ordering constraint with this grammar. This last property is actually shared by many robust parsers.

4 Discriminative LR-based parsing

The LR tables being built by preserving conflicts, determinism is achieved by a weighting component derived from the global perceptron described by (Collins, 2002). We start by describing the weighted parsing procedure before turning our attention to the weight estimation problem.

We assume that an $LR(0)$ table has been built. The GOTO function of this table GOTO: $(\Sigma \cup W) \times \mathbb{N} \mapsto \mathbb{N}$ sends a couple of symbol and LR state to a new LR state. The ACTION: $(\mathbb{N} \times W) \mapsto 2^{\mathcal{A}}$ function of this table returns a set $\mathbf{a}$ of possible actions given a state and a terminal symbol. The initial LR state of the table is σ_i while σ_e denotes a final state.

The algorithm relies on two data structures: a stack $\mathbf{S}$ and a queue. The stack $\mathbf{S} = \ldots | s_2 | s_1 | s_0$ has s_0 for topmost element. A node $s_i = \langle \sigma, \tau \rangle$ in the stack is a couple where σ is an LR state number and $\tau = (s_i.c_t[s_i.w_t] \quad s_i.c_l[s_i.w_l] \quad s_i.c_r[s_i.w_r])$ encodes a local tree of depth 1. $s_i.c_t, s_i.c_l, s_i.c_r$ denote the root left child and right child categories of tree and $s_i.w_t, s_i.w_l, s_i.w_r$ denote the root, the left child and right child terminals of this tree such that a node $s_i.c.[s_i.w.]$ denotes a non terminal 2-LCFG symbol at node s_i in the stack. The queue is static and initially filled up with the sequence of tokens to be parsed:

$$\textsc{Item} \quad \langle j, \mathbf{S} \rangle : w$$
$$\textsc{Init} \quad \langle 1, \langle \sigma_i, \epsilon \rangle \rangle : 0$$
$$\textsc{Goal} \quad \langle n+1, \langle \sigma_e, \tau \rangle \rangle : w$$

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

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

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

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

$$\text{GR} \quad \frac{\langle j, \mathbf{S}_\ominus \mid s_1 = \langle \sigma_1, (s_1.c_t[s_1.w_t] \_\_) \rangle \mid s_0 = \langle \sigma_0, (s_0.c_t[s_0.w_t] \_\_) \rangle \rangle : w_0}{\langle j, \mathbf{S}_\ominus \mid s_1 \mid \langle \textsc{Goto}(GR, \sigma_1), (s_0.c_t[s_0.w_t] \_\_) \rangle : w_0 + F(GR, \langle j, \mathbf{S} \rangle)} \qquad \text{(Rule introduced in section 5)}$$

Figure 2: Actions as inference rules in extended deductive notation

$\mathcal{T} = t_1 \dots t_n$. Parsing is performed by generating sequentially configurations $C_i = \langle j, \mathbf{S} \rangle$ where $\mathbf{S}$ is a stack and j the index of the first element of the queue. Given an initial configuration $C_0 = \langle 1, \langle \sigma_i, \epsilon \rangle \rangle$, a derivation step $C_{i-1} \overset{a_{i-1}}{\Rightarrow} C_i$ generates a new configuration $C_i = \langle j', \mathbf{S}' \rangle$ provided a configuration $C_{i-1} = \langle j, \mathbf{S}_\ominus | \langle \sigma, \tau \rangle \rangle$ by applying the action $a_{i-1} \in \textsc{Action}(\sigma, t_j)$. A $k-$step derivation sequence $C_{0 \Rightarrow k}$ is a sequence of derivation steps such that $C_0 \overset{a_0}{\Rightarrow} \dots \overset{a_{k-1}}{\Rightarrow} C_k$. A derivation sequence is finished when the configuration $C_{3n-1} = \langle n+1, \langle \sigma, \tau \rangle \rangle$ is generated[1]. If $\sigma = \sigma_e$ then the derivation is a success, otherwise it is a failure. A derivation is also finished when $\textsc{Action}(\sigma, t_j) = \emptyset$ for a configuration $C_k = \langle \sigma, t_j \rangle$ which is another case of failure. The actions detailed in Figure 2 using extended deductive notation are responsible for modifying the stack and updating LR states. The shift action, SHIFT, thus pushes onto the stack a local tree rooted by the category of the next token in the queue. The reduce left $RL(X)$ and the reduce right $RR(X)$ actions pop the top two elements from the stack and push a new element of category $X[w]$ on top of it. The two actions differ only by the way the head w is assigned: $RL(X)$ chooses w to be X's left child head word while $RR(X)$ sets w to be X's right child head word. $RU(X)$ is an unary reduction action that pops the stack top and pushes a new top element with category X whose head is its unique child head. By design of the automaton we ensure that $RU(X)$ can only be applied after a shift reduction took place.

In order to achieve disambiguation, a derivation sequence $C_{0 \Rightarrow k} = C_0 \overset{a_0}{\Rightarrow} \dots \overset{a_{k-1}}{\Rightarrow} C_k$ is also weighted by a function of the form:

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

that is the weight of a derivation sequence is given by an inner product that the parser approximates as a sum of inner products local to each derivation step. $\mathbf{w} \in \mathbb{R}^d$ is a d-dimensional vector of weights and each $\mathbf{\Phi}(a_i, C_i) \in \{0, 1\}^d$ is a d-dimensional vector of feature functions in which every ϕ_i has signature $\phi_i(a, \boldsymbol{\kappa}, j)$. The values a and j denote an action and the current index of the head of the queue in $\mathcal{T}$ while $\boldsymbol{\kappa}$ is a kernel vector similar to the one defined by (Huang and Sagae, 2010). It summarizes information accessible from the stack for the purpose of feature function evaluation. Figure 3 illustrates the actual kernel vector used in this paper: together with j, the index of the first element in the queue, the kernel

[1]For shift reduce parsing with a 2-CFG grammar, the number of steps is the number of reduction steps plus n shifts: $\eta = 2n - 1 + n = 3n - 1$.

vector $\boldsymbol{\kappa}$ is the set of values accessible to feature functions $\phi_i(a, \boldsymbol{\kappa}, j)$ in the stack. As can be seen the stack stores local trees with instanciated 2-LCFG nodes labelled with the notation introduced in section 4. Since the score of a derivation is a sum of independent terms, the (prefix) weight $w = W(C_{0\Rightarrow k})$ of a derivation sequence can be computed at each derivation step. This allows to store the (prefix) weight of this sequence on configurations such that a configuration has the extended form $C_k = \langle j, \mathbf{S}\rangle : w$ in the weighted case. We make explicit the actual prefix weight computation in Figure 2 by using the following abbreviation: $F(a_i, C_i) = \mathbf{w} \cdot \boldsymbol{\Phi}(a_i, C_i)$.

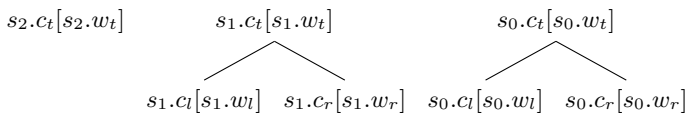


Figure 3: Representation of the kernel vector $\boldsymbol{\kappa}$

For a given input sequence $\mathcal{T}$, the parser is naturally non deterministic. Non determinism is introduced by the ACTION function which returns a set $\mathbf{a} \in 2^{\mathcal{A}}$ of possible actions given the current configuration. In the nondeterministic case, we thus derive from a given derivation sequence $C_{0\Rightarrow k-1}$ a set $\delta(C_{0\Rightarrow k-1})$ of $k-$steps derivation sequences. If we let $\text{GEN}_{k-1}(\mathcal{T})$ be the set of derivation sequences at step $k-1$, the set of derivation sequences at step k is $\text{GEN}_k(\mathcal{T}) = \bigcup_{C_{0\Rightarrow k-1}\in\text{GEN}_{k-1}(\mathcal{T})} \delta(C_{0\Rightarrow k-1})$. In this context, achieving deterministic parsing amounts to solve the following optimization problem:

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

Since in the worst case, the size of $\text{GEN}_k(\mathcal{T})$ is $|\mathcal{A}|^k$, the search space has exponential size. Like (Zhu et al., 2013), we use in this paper a beam search approximation. A beam $\text{GEN}^K_k(\mathcal{T})$ is a subset of size K of $\text{GEN}_k(\mathcal{T})$. Provided a beam $\text{GEN}^K_{k-1}(\mathcal{T})$ we build $\text{GEN}^K_k(\mathcal{T})$ with the following recurrence: $\text{GEN}^K_k(\mathcal{T}) = \text{K-argmax}_{C_{0\Rightarrow k}\in\Delta(\text{GEN}^K_{k-1}(\mathcal{T}))} W(C_{0\Rightarrow k})$ where $\Delta(\text{GEN}^K_{k-1}(\mathcal{T})) = \bigcup_{C_{0\Rightarrow k-1}\in\text{GEN}^K_{k-1}(\mathcal{T})} \delta(C_{0\Rightarrow k-1})$, Using a beam aims to reduce complexity to $\mathcal{O}(K|\mathcal{A}|(3n-1)) \approx \mathcal{O}(n)$ and makes inference computationally tractable in practice. On the other hand it makes inference incomplete (the parser may fail to find a solution even if it exists) and does not guarantee the solution to be optimal. In other words, Equation 1 is replaced by an approximation:

$$\tilde{\mathcal{C}} = \underset{C_{0\Rightarrow 3n-1}\in\text{GEN}^K_{3n-1}(\mathcal{T})}{\text{argmax}} W(C_{0\Rightarrow 3n-1}) \tag{2}$$

The weight estimation procedure is performed by the averaged percetron algorithm (Collins, 2002). As pointed out by (Huang et al., 2012) using a beam introduces an approximation that can also harm the convergence of the learning procedure since we provide at each training iteration the approximative solution given by equation 2 instead of the exact solution to equation 1 expected in theory by the perceptron algorithm. To overcome the problem we perform updates on subderivation sequences. Let $C^{(r)}_{0\Rightarrow k}$ be a subderivation sequence at step k and let $C^{(0)}_{0\Rightarrow k} = \text{argmax}_{C_{0\Rightarrow k}\in\text{GEN}^K_k(\mathcal{T})} W(C_{0\Rightarrow k})$ be the best subderivation in the beam at step k. In this context the perceptron update has the form: $\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\Phi}_g(C^{(r)}_{0\Rightarrow k}) - \boldsymbol{\Phi}_g(C^{(0)}_{0\Rightarrow k})$. We tested two methods for choosing k satisfying the weaker convergence criterions established by (Huang et al., 2012) : $C^{(0)}_{0\Rightarrow k} \neq C^{(r)}_{0\Rightarrow k}$ and $W(C^{(0)}_{0\Rightarrow k}) > W(C^{(r)}_{0\Rightarrow k})$. If we let $V = \{k \,|\, C^{(0)}_{0\Rightarrow k} \neq C^{(r)}_{0\Rightarrow k}, W(C^{(0)}_{0\Rightarrow k}) > W(C^{(r)}_{0\Rightarrow k})\}$, then the *early update* method amounts to choose $k = \min_{k\in V} k$ and the *max violation update* method amounts to choose $k = \text{argmax}_{k\in V} W(C^{(0)}_{0\Rightarrow k}) - W(C^{(r)}_{0\Rightarrow k})$.

5 Generalisations

This section introduces two extensions to the algorithm meeting practical motivations: grammar relaxation and extended word representations.

In practical cases, it may be convenient to interface the parser with a morphological tagger. In this case terminal symbols $t_1 \dots t_n$ are part of speech tags. Since grammar transformations introduced in section 2 can potentially modify the tagset and since enforcing a strict Chomsky normal form in this case makes little sense, we allow the trees to have structures such as the one given in Figure 4.

This kind of structure licences the following new patterns of 2-CFG rules: $A \rightarrow B\,t$ and $A \rightarrow t\,B$ where t denotes a terminal symbol (in this case a tag). These new rule patterns modify a property of 2-LCFG on which we relied so far, η is now variable : $n-1 \leq \eta \leq 2n-1$. We observe that longer derivation sequences tend to have an higher weight. Indeed weights increase linearly with the length of the derivation sequence as illustrated in Figure 5 where the weights are averaged out of measurements made over the parses on the French development set described in Section 6.

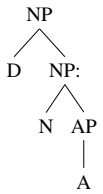


Figure 4: Relaxed tree structure

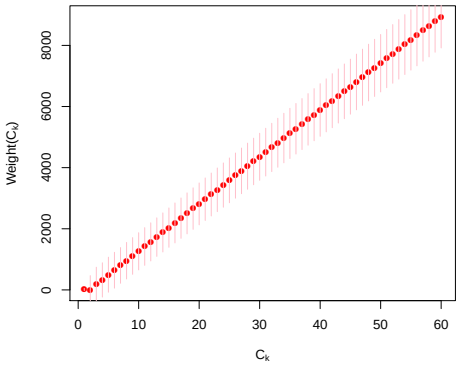


Figure 5: Average derivation weight as a function of the derivation length

Although weights can in principle be positive or negative, this apparently counter-intuitive behaviour is caused by the beam which keeps for further steps only the highest weighted configurations. To further study the behaviour of variable length sequences, we define two variants of the parser: first the 'naive' variant modifies the termination condition. Let $\mathcal{S} = \{C_{0\Rightarrow k} | C_k = \langle n+1, \langle \sigma_e, \tau \rangle \rangle, 2n-1 \leq k \leq 3n-1\}$. In this context, equation 2 is reframed as: $\tilde{\mathcal{C}} = \text{argmax}_{C_{0\Rightarrow k} \in \mathcal{S}}\ W(C_{0\Rightarrow k})$. A second version, called the 'synchronized version' introduces an additional inference rule called the Ghost Reduction and referred as GR in Figure 2. A slight modification of the LR automaton construction, designed to trigger either an unary reduction action or a Ghost reduction after a shift allow us to enforce the property that $\eta = 3n-1$ in this case too. The ghost reduction is designed to both make the parser 'wait' one step during derivation in case it chooses not to perform an unary reduction after shift and also to avoid modifying the content of the stack.

The second extension allows terminals to be not only lexical tokens or part-of-speech tags but arbitrary tuples ω. This allows to encode words with an arbitrary set of additional structured features such as their lemmas, gender, number, case, semantic representation. The exact nature of these additional features depends on the capacity of a parsing preprocessor to actually supply them. In this context the non terminal symbols of the 2-LCFG have thus the form $A[\omega]$. The fields of the tuples are then made accessible to feature functions. This extension is motivated by the hypothesis that parsing morphologically rich languages will benefit significantly from structured word representations, for instance allowing the parser to take advantage of morphology.

We are now in position to describe the feature templates used by the parser (Figure 6). Before the dot s_i and q_i denote respectively the address in the stack and in the queue of the adressed node. t, l, r denote the top, left and right nodes of the local trees in the stack. After the dot w_c, w_f denote a category and a word form, while c is a constituent category. w_m denote the mood of a verb and w_X an refined category dubbed `subcat` in the French Treebank (Abeillé et al., 2003): these subcategories refine crude tags by encoding information such as the definiteness of a determiner, subtypes of adjectives etc. gen, num, agr

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

Figure 6: Parser templates

denote the gender, the number and their interaction. The notation $e(\cdot, \cdot)$ is an equality function returning true if the values of both its argument are equal.

6 Experiments

The following experiments aim to identify the contribution of the components of the parser both to parsing accurracy and to parsing speed. Experiments are carried mainly on French. A final set of tests is also carried out on English in order to highlight the generality of the framework and to ease comparisons with other proposals.

6.1 Protocol

The experiments use the French SPMRL dataset (Seddah et al., 2013) which is newer and larger than datasets previously used for parsing French (Crabbé and Candito, 2008). It instanciates the full French Treebank described in (Abeillé et al., 2003) and will surely become the new standard data set for parsing French in the next few years. We use this data set as is, with two scenarios: one with gold standard tags and the second with tags predicted by a 97.35% accurate tagger (Seddah et al., 2013). The French data is head annotated with head rules provided by (Arun and Keller, 2005). Additionally, compound word structures are systematically left headed. For English, we use the Penn Treebank with standard split: section 02-21 for training, section 22 for development and section 23 for test. The predicted scenario uses the MELT tagger (Denis and Sagot, 2012) with an accurracy of 97.1%. The head annotations have been inferred by aligning the phrase structure treebank with its dependency conversion described by (de Marneffe et al., 2006).

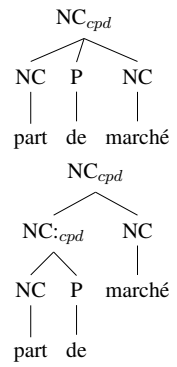
We use a C++ implementation of the algorithm described above for running the experiments. Scores reported for the Berkeley parser (Petrov et al., 2006) use the runs described by (Seddah et al., 2013). F-score is measured with the classical `evalb` and times are measured on the same machine (MacOSX 2.4Ghz) and do not take into account input/output times for both parsers.

Each experiment modifies a single experimental variable by contrast with a default parser configuration. The default parser configuration sets the beam size to $K = 4$ and uses the naive synchronisation procedure (Section 5). The LR automaton uses the grammar $G_m^{(base)}$ (Figure 7) and the update method is *early update* (Section 4). The set of templates is given in Figure 6 except for English where agreement, mood and subcat are ignored since there is no morphology directly available.

Experiment 1 This first experiment tests the impact of the beam size by running the parser with different sizes: $K = 2, K = 4, K = 8, K = 16$.

Experiment 2 The second experiment contrasts the naive synchronisation (*naive*) with the ghost reduction synchronisation (*sync*) described in section 5.

Experiment 3 This third experiment contrasts two different matrix grammars (Figure 7). This experiment aims to test whether we can take advantage of the LR automaton to better account for compound words encoded in the French data. To this end we designed two matrix grammars generating two different automata. In Figure 7, the top left tree is an example of the representation of compound words in the French data set. The corresponding binary structure is given on bottom left. The general grammar $G_m^{(base)}$ encodes a matrix grammar that does not specifically handle compound words and for which equivalence classes are $[a]$ the axiom symbol, $[n]$ non terminal symbols and $[w]$ terminal symbols. Each temporary non terminal $t \in T$ yields its own equivalence class. The grammar $G_m^{(cpd)}$ adds further equivalence classes: $[n]_{cpd}$ gathers non terminals marked as compounds (cpd) and is disjoint from $[n]$, the non compound non terminals. $T_{(cpd)}$ gathers temporary symbols marked as compounds and is disjoint from T. From these two matrix grammar we generate two different LR automata with one of them encoding a specific subgrammar for compounds (*cpd*) while the other is a generic grammar (*base*).



Immediate dominance rules for G_m^{base}		
$[a] \rightarrow [n]\,[n]$	$[n] \rightarrow [n]\,,\,[n]$	$t \rightarrow [n]\,,\,[n] \quad (\forall t \in T)$
$[a] \rightarrow [n]\,,\,[w]$	$[n] \rightarrow [n]\,,\,[w]$	$t \rightarrow [n]\,,\,[w] \quad (\forall t \in T)$
$[a] \rightarrow [w]\,[w]$	$[n] \rightarrow [w]\,,\,[w]$	$t \rightarrow [w]\,,\,[w] \quad (\forall t \in T)$
$[a] \rightarrow [w]$	$[n] \rightarrow [w]$	$t \rightarrow [n]\,,\,t \quad (\forall t \in T)$
$[a] \rightarrow [n]\,,\,t \quad (\forall t \in T)$	$[n] \rightarrow [n]\,,\,t \quad (\forall t \in T)$	$t \rightarrow [w]\,,\,t \quad (\forall t \in T)$
$[a] \rightarrow [w]\,,\,t \quad (\forall t \in T)$	$[n] \rightarrow [w]\,,\,t \quad (\forall t \in T)$	
Additional immediate dominance rules for 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: Structured representation of compound words (French data set) and related matrix grammars.

Experiment 4 This experiment contrasts *early update* with *max violation update*. Max violation update is trained over 12 epochs, while early update is trained over 25 epochs

Experiment 5 This last experiment contrasts the use of morphology. We remove (*no-morph*) the templates under mood, agreement and subcategories in Figure 6 in order to assess their impact.

6.2 Results

Results with respect to accurracy are given in table 2. Experiments are carried out on the development set with gold part of speech tags (Table 2, left) and predicted part of speech tags (Table 2, right).

Experiment 1 The parser achieves its best result with a beam of size 8 ($K = 8$). Quite surprisingly it achieves already a very correct score with a beam of size 4. We observe that increasing the size of the beam does not proide significant improvements. Using beams of size 16 (or even 32) only bring marginal accurracy improvements, if any, at the expense of more important parsing times.

Experiment 2 This second experiment is apparently more disappointing: synchronisation does not seem to play a significant role on accurracy, or a detrimental one if any. This effect seems caused by a property of the dataset. A more careful analysis of the parser error patterns shows that the parsing model naturally misses many unary reductions. Since the *naive* automaton is biased towards predicting longer sequences with higher weights, it somehow helps to favor longer derivations containing more unary reductions, hence improving the accurracy.

French dev (gold tags)				French dev (predicted tags)			
Expérience	F$\leq$ 40	F	Cov	Expérience	F$\leq$ 40	F	Cov
K=2	85.40	82.74	98.6	K=2	83.24	80.42	98.9
K=4	86.52	83.69	99.5	K=4	84.32	81.34	99.4
K=8	86.80	84.31	99.9	K=8	84.43	81.79	99.8
K=16	86.49	83.95	99.9	K=16	84.59	81.94	99.8
sync	86.41	83.66	99.6	sync	84.06	81.14	99.9
naive	86.52	83.69	99.5	naive	84.32	81.34	99.4
cpd	86.30	83.30	99.2	cpd	83.84	81.17	99.0
base	86.52	83.69	99.5	base	84.32	81.34	99.4
Max Violation	85.98	83.49	99.5	Max Violation	83.42	80.56	99.5
Early Update	86.52	83.69	99.5	Early Update	84.32	81.34	99.4
no-morph	85.23	82.43	99.8	no-morph	83.68	81.05	99.8
all-morph	86.52	83.69	99.5	all-morph	84.32	81.34	99.4

Table 2: Experimental results (development)

Experiment 3 This experiment highlights the problems related to further constraining a parser with approximative search: we observe that parsing coverage is reduced. This can be explained by the fact that further constraining the grammar creates less success states in the automaton and that the parser sometimes has to perform less local decisions without all the necessary information available. This suggests for further work that a more constrained matrix grammar should be used with a more robust search strategy than simple beam search.

Experiment 4 In experiment 4, we observe that max violation update converges twice as fast as early update but we experienced more overfitting problems explaining the lower scores. It is however harder to achieve fair comparisons since the number of iterations is significantly different.

Experiment 5 This last experiment is probably the most significative. We observe that morphology is the variable that allows the parser to improve significantly on French (dev F=81.79). This result thus confirms those observed by (Hohensee and Bender, 2012) on several languages for dependency parsing, yet we had to isolate agreement, refined subcategories and verbal mood to get significant improvements (Figure 6).

Final tests In order to compare this proposal with current state of the art parsers, we provide comparative measures of speed and accurracy with the Berkeley parser (Petrov et al., 2006), known to be representative of the state of the art in accurracy and in speed on French and in accurracy on English (Table 3).

French Test (gold tags)	F$\leq$ 40	F	Cov
K=8	87.14	84.20	99.8
Berkeley	86.44	83.96	99.9
French Test (predicted tags)	F$\leq$ 40	F	Cov
K=8	84.33	81.43	99.8
Berkeley	83.16	80.73	99.9
French test (raw text)	F$\leq$ 40	F	Cov
Berkeley	83.59	81.33	99.9
English test (gold tags)	F$\leq$ 40	F	Cov
K=8	90.2	89.5	100
English test (pred tags)	F$\leq$ 40	F	Cov
K=8	89.7	89.1	100
English test (raw text)	F$\leq$ 40	F	Cov
Berkeley	-	90.1	-

Table 3: Experimental results (test)

Interestingly parsing the English dataset with templates designed on French data is almost state of the art on English (dev F=89.3, test F=89.1). This suggests that feature engineering is a less important issue than often thought and it also suggests that the parser is likely to be easy to adapt to other languages by generalizing the method used to parse English.

Although the differences are modest, the parser is state of the art on French (F=81.43) if we compare with the Berkeley parser (F=80.73) known to be indicative of the state of the art on French (Seddah et al., 2013) and if we ignore ensemble and semi-supervised parsers.

The most important difference is related to speed. (Petrov et al., 2006) is reported to be the fastest phrase structure parser for English by (Huang and Sagae, 2010) where the authors compare with (Charniak, 2000). Yet (Petrov et al., 2006) is a polynomial time parser, while this one has linear time behaviour. We compared the speed of both parsers on the same hardware (ignoring input/output times) and we find (Petrov et al., 2006) has an average parse time of $t_\mu = 0.28$s with maximum $t_{max} = 10.27$s while our linear time algorithm has mean time $t_\mu = 0.06$s and maximum $t_{max} = 0.1$s with beam 4 and $t_\mu = 0.1s, t_{max} = 0.5s$ with beam 8 (Figure 8). Further constraining the automaton shows to be useful for speed, since for experiment 3, $t_\mu = 0.04s$ with $K = 4$ which is clearly faster.

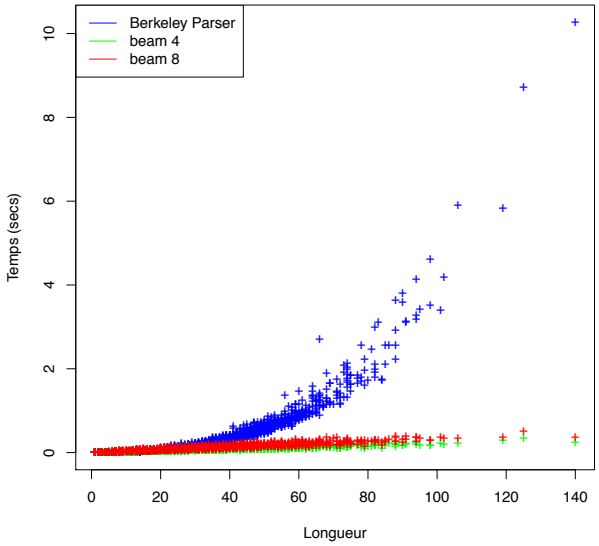


Figure 8: Parsing times

We finally observe on English that the LR strategy performs reasonably accurately and faster than the implementation of (Petrov et al., 2006) whereas optimisations of PCFG-LA described by (Bodenstab et al., 2011) are significantly less accurate and not significantly faster. Although it is currently hard to to compare directly on French with the most similar proposal, the Chinese/English parser described by (Zhu et al., 2013), since it does not handle morphology. With respects to speed, their impressive result on English suggests that there is still room for speed improvements without loss of accurracy.

7 Conclusion

To our knowledge, this is the first formulation of a discriminative LR-automaton driven parser for natural language. This LR inspired algorithm shares many properties with shift reduce parsers for phrase structure grammar described by (Sagae and Lavie, 2006) and (Zhu et al., 2013). (Sagae and Lavie, 2006) describe a first version of this kind of algorithm using a weighting system based on a local maximum entropy classifier. It thus enables them to use a best first search strategy that allows in principle to achieve near-optimal parsing. By contrast, like (Zhu et al., 2013), we use here a global perceptron algorithm together with a beam based breadth first search to which we add an explicit LR component. The LR automaton allows us to guarantee that the parser generates a viable prefix. We believe the LR framework can also shed light on theoretical, practical and experimental issues related to phrase structure parsing by comparison with dependency parsing. However using the LR automaton to constrain the parsing model

for multi-word expressions turns out to be disappointing since it forces the parser to take less local decisions for which the beam approximation is not well suited. This suggests for future work to explore search methods aiming to achieve optimality (Zhao et al., 2013).

Mirroring a common practice in dependency parsing, the parser also provides a first support for phrase structure parsing of morphologically rich languages thanks to structured word representations. The richer lexical structure makes morphological information available during the parsing process. For the case of French it enables, among others, to integrate agreement in the parsing model. This simple integration of morphology then allows the parser to achieve state of the art accurracy on French. Since in principle nothing in the algorithm is specific to French, we expect to generalize and experiment with the model on other morphologically rich languages. Further work for such languages is expected to involve a refinement of the interface with morphology along the lines of (Hatori et al., 2012; Bohnet et al., 2013).

Acknowledgements

I am grateful to Maximin Coavoux who helped at some stages of the implementation. I am also grateful to Benoit Sagot for his encouragments and several discussions that helped to clarify the contents of this paper, and finally to Djamé Seddah who brought his expertise with the actual data sets.

References

Anne Abeillé, L. Clément, and F. Toussenel. 2003. Building a treebank for french. In *Treebanks*. Kluwer.

Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, and Monica S. Lam. 2006. *Compilers: Principles, Techniques, and Tools*. Addison Wesley.

Abhishek Arun and Frank Keller. 2005. Lexicalization in crosslinguistic probabilistic parsing: The case of french. In *Association for Computational Linguistics*.

Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark. 2011. Beam-width prediction for efficient context-free parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Portland, Oregon, June. Association for Computational Linguistics.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajic. 2013. Joint morphological and syntactic analysis for richly inflected languages. *TACL*, 1:415–428.

Ted Briscoe and John A. Carroll. 1993. Generalized probabilistic lr parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *North American Association for Computational Linguistics*.

Michael Collins. 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)*.

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

Benoit Crabbé and Marie Candito. 2008. Expériences d'analyses syntaxique statistique du français. In *Actes deTALN 2008*.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning . In LREC 2006. 2006. Generating typed dependency parses from phrase structure parses. In *Language ressources and evaluation conference*.

Pascal Denis and Benoît Sagot. 2012. Coupling an annotated corpus and a lexicon for state-of-the-art pos tagging. *Language Resources and Evaluation*, 46(1).

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of the Association for Computational Linguistics*.

Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized Phrase Structure Grammar*. Cambridge, MA: Harvard University Press and Oxford: Basil Blackwell's.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2012. Incremental joint approach to word segmentation, pos tagging, and dependency parsing in chinese. In *ACL (1)*, pages 1045–1053.

Matt Hohensee and Emily M. Bender. 2012. Getting more from morphology in multilingual dependency parsing. In *HLT-NAACL*, pages 315–326.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *North American Association for Computational Linguistics*.

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

Mark-Jan Nederhof and Giorgio Satta. 2010. Algorithmic aspects of natural language processing. In M.J. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook*. CRC press.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 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.

Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL*.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Wolińsk, Alina Wróblewska, and Éric Villemonte De La Clergerie. 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*.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Conference on Empirical Methods in Natural Language Processing*.

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

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

Joseph P. Turian and I. Dan Melamed. 2006. Advances in discriminative parsing. In *ACL*.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1).

Kai Zhao, James Cross, and Liang Huang. 2013. Optimal incremental parsing via best-first dynamic programming. In *EMNLP*, pages 758–768.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Association for Computational Linguistics*.