HAL
open science

Integrating Selectional Constraints and Subcategorization Frames in a Dependency Parser

Seyed Abolghasem Mirroshandel, Alexis Nasr

▶ **To cite this version:**

Seyed Abolghasem Mirroshandel, Alexis Nasr. Integrating Selectional Constraints and Subcategorization Frames in a Dependency Parser. Computational Linguistics, 2016, 42, pp.55-90. 10.1162/COLI_a_00242 . hal-01454761

HAL Id: hal-01454761

https://hal.science/hal-01454761v1

Submitted on 3 Feb 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Integrating Selectional Constraints and Subcategorization Frames in a Dependency Parser

Seyed Abolghasem Mirroshandel*
University of Guilan

Alexis Nasr**
Université Aix-Marseille

Statistical parsers are trained on treebanks that are composed of a few thousand sentences. In order to prevent data sparseness and computational complexity, such parsers make strong independence hypotheses on the decisions that are made to build a syntactic tree. These independence hypotheses yield a decomposition of the syntactic structures into small pieces, which in turn prevent the parser from adequately modeling many lexico-syntactic phenomena like selectional constraints and subcategorization frames. Additionally, treebanks are several orders of magnitude too small to observe many lexico-syntactic regularities, such as selectional constraints and subcategorization frames. In this article, we propose a solution to both problems: how to account for patterns that exceed the size of the pieces that are modeled in the parser and how to obtain subcategorization frames and selectional constraints from raw corpora and incorporate them in the parsing process. The method proposed was evaluated on French and on English. The experiments on French showed a decrease of 41.6% *of selectional constraint violations and a decrease of* 22% *of erroneous subcategorization frame assignment. These figures are lower for English:* 16.21% *in the first case and* 8.83% *in the second.*

1. Introduction

The fundamental problem we address in this article was formulated by Joshi (1985) as the following question: *How much context-sensitivity is required to provide reasonable structural descriptions?* His answer to this question was the extended domain of locality principle of Tree Adjoining Grammars, which allows us to represent in a single structure (an elementary tree[1]) a predicate and its arguments.

* Computer Engineering Department, Faculty of Engineering, University of Guilan, Rasht, Iran.
E-mail: mirroshandel@guilan.ac.ir.

** Laboratoire d'Informatique Fondamentale, CNRS - Université Aix-Marseille, France.
E-mail: alexis.nasr@univ-amu.fr.

1 In the remainder of this article, we will refer to the elementary object manipulated by the parser as **factors**, borrowing this term from graph-based parsing.

Submission received: 10 September 2014; revised version received: 3 August 2015; accepted for publication: 28 September 2015.

doi:10.1162/COLI_a_00242

© 2016 Association for Computational Linguistics

Context sensitivity and the correct size of domain of locality for a reasonable structural description has always been a major issue for syntactic formalisms and parsers. Giving an overview of the different solutions that have been proposed to answer this question is clearly beyond the scope of this article. We will limit our discussion to the framework of graph-based dependency parsing (Eisner 1996; McDonald and Pereira 2006), in which our study takes place. The basic first-order model makes an extreme choice with respect to the domain of locality by limiting it to one dependency (the score of a tree is the sum of the scores of its dependencies). Second-order models slightly extend the factor sizes to second-order patterns that include a dependency adjacent to the target dependency. It has been shown experimentally that second-order models perform better than first-order ones, which tends to prove that second-order factors do capture important syntactic regularities. More recently, Koo and Collins (2010) proposed extending factor size to three dependencies and showed that such models yield better accuracy than second-order models on the same data. However, extending the order of the models has a high computational cost because the number of factors to be considered in a sentence grows exponentially with the order of the model. Using such models in a Dynamic Programming framework, such as Eisner (1996), becomes quickly intractable in terms of processing time and memory requirements. Additionally, by reliably estimating the scores of such factors we are quickly confronted with the problem of data sparseness.

One can also note that high-order factors are not needed for all syntactic phenomena. In fact, most syntactic attachments can be accurately described with first- and second-order models. This is why first- and second-order models perform quite well: They reach a labeled accuracy score of 85.36% and 88.88% for first- and second-order models, respectively, on the French Treebank (Abeillé, Clément, and Toussenel 2003). The same models trained on the Penn Treebank (Marcus, Marcinkiewicz, and Santorini 1993) reach 88.57% and 91.76% labeled accuracy score, respectively. The extension of the domain of locality can therefore be limited to some syntactic phenomena. This is the case in Tree Adjoining Grammars for which the extended domain of locality only concerned some aspects of syntax, namely, subcategorization.

The solution we explore in this article, in order to take high-order factors into account in a parser, is to decompose the parsing process into two sub-processes. One of them is in charge of local syntactic phenomena and does not need a high-order model and the other takes care of syntactic phenomena that are beyond the scope of the first one. We will call the first one **parsing** and the second one **patching**. The patching process is responsible for modeling two important aspects of syntax, Selectional Constraints (SCs) and Subcategorization Frames (SFs), which are usually poorly modeled in parsers, as we will show in Sections 7 and 8. Parsing and patching differ in two important aspects. First, they rely on different search algorithms. The first one is based on Dynamic Programming and the other one uses Integer Linear Programming (ILP). Second, they rely on different data sets. The first uses a standard treebank whereas the second uses much larger unannotated corpora.

The reason for the first difference comes from the different nature of these two processes. As already mentioned, high-order factors are needed to model some specific syntactic phenomena and we do not want to systematically increase the order of the model in order to take them into account. Using factors of different sizes in a Dynamic Programming parser is not practical and leads to ad hoc adaptation of the parser. This is the reason why patching is based on ILP, which can accommodate more naturally for constraints defined on structures of different sizes.

The second difference between parsing and patching is the data sets that were used to train them. The parser is trained on a standard treebank whereas the patching model is trained on a raw corpus that is several orders of magnitude larger. The reason for this difference comes from the fact that high-order factors used to model SFs and SCs contain up to three lexical elements and cannot be accurately modeled using available treebanks. It has been shown by Gildea (2001) and Bikel (2004) that bilexical dependencies of the Collins parser (Collins 1997) have a very limited impact on the performances of the parser, although they were supposed to play a key role for some important ambiguous syntactic attachments. The reason for this is that treebanks are not large enough to correctly model bilexical dependencies. We show in Sections 7.2 and 8.2 that current treebanks are much too limited in size to accurately model SCs and SFs.

Other solutions have been proposed in the literature in order to combine local and global syntactic phenomena in a parser. Parse reranking is one of them. The idea is to produce the n-best parses for a sentence and rerank them using higher order features, such as in Collins and Koo (2005) and Charniak and Johnson (2005). This solution has the advantage of drastically limiting the search space of the high-order search algorithm to the k best parses. Although the parsing architecture we propose in this article does use k-best parse lists, our solution allows us to combine dependencies that appear in any of the parses of the list. We show in Section 6.3 that combining dependencies that appear in any parse of a k-best list can yield parses that are more accurate than the best parse of this k-best list. Our method is closer to forest rescoring (Huang and Chiang 2007) or forest reranking (Huang 2008).

Another solution to combine local and global decisions is to represent the local and the global constraints as a single ILP program. This solution is possible because dependency parsing can be framed as an ILP program. Riedel and Clarke (2006) propose a formulation of nonprojective dependency parsing as an ILP program. This program, however, requires an exponential number of variables. Martins, Smith, and Xing (2009) propose a more concise formulation that only requires a polynomial number of variables and constraints. Martins, Almeida, and Smith (2013) propose extending the model to third-order factors and using dual decomposition. All these approaches have in common their ability to produce nonprojective structures (Lecerf 1961). Our work departs from these approaches in that it uses a dynamic programming parser combined with an ILP solver. Additionally, we do not take nonprojective structures into account.

The structure of this article is as follows. In Section 2 we describe the type of parser used in this work. Section 3 describes the patching process that detects, in a given sentence, a set of optimal SFs and SCs, using ILP. The exact nature of SFs and SCs is described in that section. Section 4 proposes a way to combine the solutions produced by the two methods using constrained parsing. In Section 5, we justify the use of ILP by showing that the patching process is actually NP-complete. Sections 6, 7, and 8 constitute the experimental part of this work. Section 6 describes the experimental set-up, and Sections 7 and 8, respectively, give results obtained on French and English. Section 9 concludes the article.

The material presented in this article has been partially published in Mirroshandel, Nasr, and Roux (2012) and Mirroshandel, Nasr, and Sagot (2013). In Mirroshandel, Nasr, and Roux (2012), a method for taking into account selectional constraints in a parser is presented, and in Mirroshandel, Nasr, and Sagot (2013) a method of introducing subcategorization frames in the parser is described. This paper proposes a general framework for dealing with these two problems, both separately and jointly, using Integer Linear Programming, whereas the two previous techniques used ad hoc adaptations

and could not treat the two problems jointly. Additionally, we extended the experimental part of our work to English, whereas previous work only concerned French.

2. Parsing

In this section, we briefly present the graph-based approach for projective dependency parsing. We then introduce an extension to this kind of parser, which we call **constrained parsing**, that forces the parser to include some specific patterns in its output. Then, a simple confidence measure that indicates how confident the parser is about any subpart of the solution is presented.

2.1 Graph-Based Parsing

Graph-based parsing (McDonald, Crammer, and Pereira 2005a; Kübler, McDonald, and Nivre 2009) defines a framework for parsing that does not make use of a generative grammar. In such a framework, given a sentence $S = w_1 \dots w_l$, any dependency tree[2] for S is a possible syntactic structure for S. The heart of a graph-based parser is the scoring function, which assigns a score $s(T)$ to every tree $T \in \mathcal{T}(S)$, where $\mathcal{T}(S)$ denotes the set of all dependency trees of sentence S. Such scores are usually the sum of the scores of subparts of S. The parser returns the tree that maximizes such a score:

$$\hat{T} = \underset{T \in \mathcal{T}(S)}{\arg\max} \sum_{\psi \in \psi(T)} s(\psi) \tag{1}$$

In Equation (1), $\psi(T)$ is the set of all the relevant subparts of tree T and $s(\psi)$ is the score of subpart ψ. The scores of the subparts are computed using machine learning algorithms on a treebank, such as McDonald, Crammer, and Pereira (2005b).

As already mentioned in Section 1, several decompositions of the tree into subparts are possible and yield models of increasing complexity. The most simple one is the **first-order model**, which simply decomposes a tree into single dependencies and assigns a score to a dependency, irrespective of its context. The more popular model is the **second-order model**, which decomposes a tree into subparts made of two dependencies.

Solving Equation (1) cannot be done naively for combinatorial reasons: The set $\mathcal{T}(S)$ contains an exponential number of trees. However, it can be solved in polynomial time using dynamic programming (Eisner 2000).

2.2 Constrained Parsing

The graph-based framework allows us to impose structural constraints on the parses that are produced by the parser in a straightforward way, a feature that will prove to be valuable in this work.

A sentence $S = w_1 \dots w_l$ is a parser with a scoring function s and a dependency $d = (i, r, j)$, such that i is the position of the governor, j the position of the dependent,

2 A dependency tree for sentence $S = w_1 \dots w_l$ and the dependency relation set $\mathcal{R}$ is a directed labeled tree (V, A) such that $V = \{w_1, \dots w_n\}$ and $A \subseteq V \times \mathcal{R} \times V$.

and r the dependency label, with $1 \leq i, j \leq l$. We can define a new scoring function s_d^+ in the following way:

$$s_d^+(i', r', j') = \begin{cases} -\infty & \text{if } j' = j \text{ and} \\ & (i' \neq i \text{ or } r' \neq r) \\ s(i', r', j') & \text{otherwise} \end{cases}$$

When running the parser on sentence S, with the scoring function s_d^+, one can be sure that dependency d will be part of the solution produced by the parser because the score of all its competing dependencies (i.e., dependencies of the form (i', r', j') with different governors ($i' \neq i$) or labels ($r' \neq r$) and the same dependent ($j' = j$)) are set to $-\infty$. As a result, the overall score of trees that have such competing dependencies cannot be higher than the trees that have our desired dependency. In other words, the resulting parse tree will contain dependency d. This feature will be used in Section 4 in order to parse a sentence for which a certain number of dependencies are set in advance.

Function s_d^+ can be extended in order to force the presence of any set of dependencies in the output of the parser. Suppose $\mathcal{D}$ is a set of dependencies; we then define $s_{\mathcal{D}}^+$ the following way:

$$s_{\mathcal{D}}^+(i, r, j) = \begin{cases} s_d^+(i, r, j) \text{ if } \exists d \in \mathcal{D} \text{ such that } d = (\cdot, \cdot, j) \\ \\ s(i, r, j) \quad \text{otherwise} \end{cases}$$

Modifying the scoring function of the parser to force the inclusion or the exclusion of some dependencies has already been used in the literature. It was used by Koo and Collins (2010) to ignore dependencies whose probability was below a certain threshold and by Rush and Petrov (2012) in their multi-pass, coarse-to-fine approach.

2.3 Confidence Measure

The other extension of graph-based parsers that we use in this work is the computation of confidence measures on the output of the parser. The confidence measure used is based on the estimation of posterior probability of a subtree, computed on the k-best list of parses of a sentence. It is very close to confidence measures used for speech recognition (Wessel et al. 2001) and differs from confidence measures used for parsing, such as Sánchez-Sáez, Sánchez, and Benedí (2009) or Hwa (2004), which compute confidence measures for complete parses of a sentence.

Given the k-best parses of a sentence and a subtree Δ present in at least one of the k-best parses, let $\mathcal{C}(\Delta)$ be the number of occurrences of Δ in the k-best parse set. Then $CM(\Delta)$, the confidence measure associated with Δ, is computed as:

$$CM(\Delta) = \frac{\mathcal{C}(\Delta)}{k}$$

Mirroshandel and Nasr (2011) and Mirroshandel, Nasr, and Roux (2012) provide more detail on the performance of this measure. Other confidence measures could be used, such as the edge expectation described in McDonald and Satta (2007), or inside-outside probabilities.

3. Patching

In this section, we introduce the process of *patching* a sentence. The general idea of patching sentence S is to detect in S the occurrences of predefined configurations. In section 3.1 we introduce Lexico-Syntactic Configurations, which correspond to the *patches*. We describe in section 3.2 the process of selecting an optimal set of patches on a sentence (finding an optimal solution is not trivial since patches have scores and some patches are incompatible). Sections 3.3 and 3.4, respectively, describe two instances of the patching problem, namely, subcategorization frame selection and selectional constraint satisfaction. Section 3.5 shows how both problems can be solved together.

3.1 Lexico-Syntactic Configuration

A Lexico-syntactic Configuration (LSC) is a pair (s, T) where s is a numerical value, called the **score** of the LSC, and T is a dependency tree of arbitrary size. Every node of the dependency tree has five features: a syntactic function, a part of speech, a lemma, a fledged form, and an index, which represents a position in a sentence. The features of every node can either be specified or left unspecified. Given an LSC i, $s(i)$ will represent the score of i and $T(i)$ will represent its tree.

Index features are always unspecified in an LSC. What follows is an example of an LSC that corresponds to the sentence *Jean offre des fleurs à N* [*Jean offers flowers to N*]. The five features of the dependency tree nodes are represented between square brackets and are separated by colons.

```
(0.028, [:V:offrir:offre:](
          [SUJ:N:Jean:Jean:],
          [OBJ:N:fleur:fleurs:],
          [A-OBJ:P:a:a:](
            [OBJ:N:::])))
```

An instantiated LSC (ILSC) is an LSC such that all its node features (in particular, index features) are specified. We will note $R(i)$ as the index of the root of ILSC i and $\mathcal{L}(i)$ as the set of its leaf indices.

Given the sentence *Jean offre des fleurs à Marie* and the LSC, the instantiation of the LSC on the sentence gives the following ILSC:

```
(0.028, [:V:offrir:offre:2](
          [SUJ:N:Jean:Jean:1],
          [OBJ:N:fleur:fleurs:4],
          [A-OBJ:P:a:a:5](
            [OBJ:N:Marie:Marie:6])))
```

3.2 Selecting the Optimal Set of ILSCs

Given a sentence S, and a set $\mathcal{L}$ of LSCs, we define the set $\mathcal{I}$ of all possible instantiations of elements of $\mathcal{L}$ on S. Once the set $\mathcal{I}$ has been defined, our aim is to select a subset $\hat{\mathcal{I}}$ made of compatible[3] ILSCs such that

$$\hat{\mathcal{I}} = \arg\max_{I \subseteq \mathcal{I}} \sum_{C \in I} s(C) \tag{2}$$

As it was the case for Equation (1), Equation (2) cannot be solved naively for combinatorial reasons: The power-set of $\mathcal{I}$ contains an exponential number of elements. Such a problem will be solved using ILP.

In order to solve Equation (2), the set $\mathcal{I}$ of all possible instantiations of the LSCs of set $\mathcal{L}$ for a given sentence S must be built. Several methods can be used to compute such a set, as described in Section 6.3.

We will see in Sections 3.3 and 3.4 two instances of the patching problem. In Section 3.3, ILSCs will represent SFs, and in Section 3.4, they will represent SCs.

3.3 Subcategorization Frames

A subcategorization frame is a special kind of LSC. Its root is a predicate and its leaves are the arguments of the predicate.

Here is an example of an SF for the French verb *donner* [to give]. It is a ditransitive frame; it has both a direct object and an indirect one introduced by the preposition *à* as in *Jean donne un livre à Marie.* [Jean gives a book to *Marie*].

```
(0.028, [:V:donner::] (
          [SUJ:N:::],
          [OBJ:N:::],
          [A-OBJ:P:a:a:] (
            [OBJ:N:::])))
```

The score of an SF T must reflect the tendency of a verb V (the root of the SF) to select the given SF. This score s_{SF} is simply the conditional probability $P(T|V)$ estimated data sets that will be described in Sections 7 and 8.

3.3.1 Selecting the Optimal Set of Subcategorization Frames. Given a sentence S of length N and the set $\mathcal{I}$ of Instantiated Subcategorization Frames (ISFs) over S, the selection of the optimal set $\hat{\mathcal{I}} \subseteq \mathcal{I}$ is done by solving the general equation 2, using ILP, with constraints that are specific to SF selection. The exact formulation of the ILP program is now described.

- *Definition of the variables*
  - $\alpha_i^j = 1$ if word number i is the predicate of ISF number j, and $\alpha_i^j = 0$ otherwise.

3 We will not say more at this point about the compatibility of two ILSCs. The exact definition of incompatibility will be given in Sections 3.3 and 3.4.

- $\beta_i^j = 1$ if word number i is an argument of ISF number j, and $\beta_i^j = 0$ otherwise.
  Variables α_i^j and β_i^j are not defined for all values of i and j. A variable α_i^j is defined only if word i can be the predicate of ISF $j \in \mathcal{I}$.

- *Definition of the constraints*
  - a word is the predicate of at most one ISF:

$$\forall i \in \{1, \ldots, N\} \sum_{j \in \mathcal{I}} \alpha_i^j \leq 1$$

  - a word cannot be an argument of more than one ISF:

$$\forall i \in \{1, \ldots, N\} \sum_{j \in \mathcal{I}} \beta_i^j \leq 1$$

  - for an ISF to be selected, both its predicate and all its arguments (i.e., $\mathcal{L}(j)$) must be selected:

$$\forall j \in \{1, \ldots, |\mathcal{I}|\} \; |\mathcal{L}(j)| \alpha_{R(j)}^j - \sum_{l \in \mathcal{L}(j)} \beta_l^j = 0$$

- *Definition of the objective function*

$$\max \sum_{j \in \mathcal{I}} \alpha_{R(j)}^j s_{SF}(j)$$

Let us illustrate this process on the sentence *Jean rend le livre qu'il a emprunté à la bibliothèque.* [*Jean returns the book that he has borrowed from the library.*] which contains an ambiguous prepositional phrase attachment: the prepositional phrase *à la bibliothèque* [*from the library*] can be attached either to the verb *rendre* [*return*] or to the verb *emprunter* [*borrow*] . The set $\mathcal{I}$ is composed of the four following ISFs:

```
1 (0.2,[:V:rend:rendre:2](
      [SUJ:N:Jean:Jean:1],
      [OBJ:N:livre:livre:4]))

2 (0.4,[:V:rend:rendre:2](
      [SUJ:N:Jean:Jean:1],
      [OBJ:N:livre:livre:4],
      [A-OBJ:P:a:a:9](
        [OBJ:N:biblio.:biblio.:11])))

3 (0.3,[:V:emprunte:emprunter:8] (
      [SUJ:N:il:il:6],
      [OBJ:N:qu':que:5]))

4 (0.6,[:V:emprunte:emprunter:8] (
      [SUJ:N:il:il:6],
      [OBJ:N:qu':que:5],
      [A-OBJ:P:a:a:9] (
        [OBJ:N:biblio.:biblio.:11])))
```

The actual ILP program is the following:

- *Definition of the variables*

α_2^1, α_2^2	verb *rendre* can select ISF 1 or 2
α_8^3, α_8^4	verb *emprunter* can select ISF 3 or 4
β_1^1, β_1^2	noun *Jean* can be the subject of ISF 1 or 2
β_4^1, β_4^2	noun *livre* can be the object of ISF 1 or 2
β_{11}^1	noun *bibliotheque* can be the indirect object of ISF 2
β_5^3, β_5^4	relative pronoun *que* can be the object of ISF 3 or 4
β_6^3, β_6^4	pronoun *il* can be the subject of ISF 3 or 4
β_{11}^4	noun *bibliotheque* can be the indirect object of ISF 4

- *Definition of the constraints*
  - a word is the predicate of at most one ISF:

$\alpha_2^1 + \alpha_2^2 \leq 1$	verb *rendre* cannot be the predicate of both ISF 1 and 2
$\alpha_8^3 + \alpha_8^4 \leq 1$	verb *emprunter* cannot be the predicate of both ISF 3 and 4

  - a word cannot be an argument of more than one ISF:

$\beta_1^1 + \beta_1^2 \leq 1$	Noun *Jean* cannot be the subject of both ISF 1 and 2
$\beta_4^1 + \beta_4^2 \leq 1$	Noun *livre* cannot be the object of both ISF 1 and 2
$\beta_5^3 + \beta_5^4 \leq 1$	Relative pronoun *que* cannot be the object of both ISF 3 and 4
$\beta_6^3 + \beta_6^4 \leq 1$	Pronoun *il* cannot be the subject of both ISF 3 and 4
$\beta_{11}^2 + \beta_{11}^4 \leq 1$	Noun *bibliotheque* cannot be the object of both ISF 2 and 4

  - for an ISF to be selected, both its predicate and all its arguments must be selected:

$2\alpha_2^1 - (\beta_1^1 + \beta_4^1) = 0$	ISF 1 must have a subject and an object
$3\alpha_2^2 - (\beta_1^2 + \beta_4^2 + \beta_{11}^2) = 0$	ISF 2 must have a subject, an object and an indirect object
$2\alpha_8^3 - (\beta_5^3 + \beta_6^3) = 0$	ISF 3 must have a subject and an object
$3\alpha_8^4 - (\beta_5^4 + \beta_6^4 + \beta_{11}^4) = 0$	ISF 4 must have a subject, an object and an indirect object

- *Definition of the objective function*

$$\max(0.2\alpha_2^1 + 0.4\alpha_2^2 + 0.3\alpha_8^3 + 0.6\alpha_8^4)$$

The problem admits eight solutions, represented in Figure 1. Each column corresponds to a variable. Black dots (•) correspond to the value 1 and white dots (○) to the value 0. The optimal solution is solution number 7 for which verb *rendre* selects ISF 1 and verb *emprunter* selects ISF 4. In this solution, the prepositional phrase *à la bibliothèque* is attached to the verb *rendre*.

3.4 Selectional Constraints

A selectional constraint (SC) is another kind of LSC. Its dependency tree is either made of a single dependency or two dependencies in a (daughter, mother, grandmother) configuration. An SC models the tendency of two words (the root and the leaf) to co-occur in a specific syntactic configuration.

	α^1_2	β^1_1	β^1_4	α^2_2	β^2_1	β^2_4	β^2_{11}	α^3_8	β^3_5	β^3_6	α^4_8	β^4_5	β^4_6	β^4_{11}	*score*
1	○	○	○	○	○	○	○	○	○	○	○	○	○	○	0.0
2	●	●	●	○	○	○	○	○	○	○	○	○	○	○	0.2
3	○	○	○	●	●	●	●	○	○	○	○	○	○	○	0.4
4	○	○	○	○	○	○	○	●	●	●	○	○	○	○	0.3
5	○	○	○	○	○	○	○	○	○	○	●	●	●	●	0.6
6	●	●	●	○	○	○	○	●	●	●	○	○	○	○	0.5
7	●	●	●	○	○	○	○	○	○	○	●	●	●	●	**0.8**
8	○	○	○	●	●	●	●	●	●	●	○	○	○	○	0.7

Figure 1
Representation of the eight solutions to the ILP problem.

A set of four SC patterns, described here, have been defined. The first two describe a subject and object selectional constraint, respectively. Patterns 3 and 4, respectively, describe selectional constraints on an indirect object introduced by the preposition *de* and *à* :

```
([:V:::]([SUJ:N:::]))
([:V:::]([OBJ:N:::]))
([:V:::]([DE-OBJ:P:de::]([OBJ:N:::])))
([:V:::]([A-OBJ:P:a::]([OBJ:N:::])))
```

Three important formal features distinguish SFs and SCs:

- The domain of locality of SFs usually exceeds the domain of locality of SCs.
- SCs model bilexical phenomena (the tendency of two words to co-occur in a specific syntactic configuration) whereas SFs model unilexical phenomena (the tendency of one word to select a specific SF).
- The number of different SC patterns is set a priori whereas the number of SF patterns is data driven.

Note that SC patterns are described at a surface syntactic level and, in the case of control and raising verbs, the SC pattern will be defined over the subject and the control/raising verb and not the embedded verb—although the selectional constraint is between the subject and the embedded verb. In the event of coordination of subject or object, only one of the coordinated elements will be extracted.

The score of an SC should reflect the tendency of the root lemma l_r and the leaf lemma l_l to appear together in configuration C. It should be maximal if whenever l_r occurs as the root of configuration C, the leaf position is occupied by l_l and, symmetrically, if whenever l_l occurs as the leaf of configuration C, the root position is occupied by l_r. A function that conforms to such a behavior is the following:

$$s_{SC}(C, l_r, l_l) = \frac{1}{2}\left(\frac{\mathcal{C}(C, l_r, l_l)}{\mathcal{C}(C, l_r, *)} + \frac{\mathcal{C}(C, l_r, l_l)}{\mathcal{C}(C, *, l_l)}\right)$$

where $\mathcal{C}(C, l, *)$ (respectively, $\mathcal{C}(C, *, l)$) is the number of occurrences of configuration C with lemma l as a root (respectively, leaf) and $\mathcal{C}(C, l_r, l_l)$ is the number of occurrences of configuration C with lemma l_r as a root and lemma l_l as a leaf, simultaneously.

This function takes its values between 0 (l_r and l_l never co-occur) and 1 (l_r and l_l always co-occur). It is close to pointwise mutual information (Church and Hanks 1990) but takes its values between 0 and 1.

3.4.1 Selecting the Optimal Set of Selectional Constraints. Given a sentence S of length N and the set $\mathcal{I}'$ of Instantiated Selectional Constraints (ISCs) over S, the selection of the optimal set $\hat{\mathcal{I}}' \subseteq \mathcal{I}'$ is framed as the following ILP program.

- *Definition of the variables*
  - $\gamma_i^j = 1$ if word number i is the root of ISC number j, and $\gamma_i^j = 0$ otherwise.
  - δ_i^j if word number i is the leaf of ISC number j, and $\delta_i^j = 0$ otherwise.
- *Definition of the constraints*
  - a word cannot be the leaf of more than one ISC

$$\forall i \in \{1, \ldots N\} \sum_{j \in \mathcal{I}'} \delta_i^j \leq 1$$

  - for an ISC j to be selected, both its root (i.e., $R(j)$) and its leaf must be selected

$$\forall j \in \{1, \ldots, |\mathcal{I}'|\} \; \gamma_{R(j)}^j - \delta_{d \in \mathcal{L}(j)}^j = 0$$

- *Definition of the objective function*

$$\max \sum_{j \in \mathcal{I}'} \gamma_{R(j)}^j s_{SC}(j)$$

Let us illustrate the selection process on the same sentence (*Jean rend le livre qu'il a emprunté à la bibliothèque.*). A set of four ISCs is defined:

```
5 (0.2,[:V:rend:rendre:2]
       ([SUJ:N:Jean:Jean:1]))
6 (0.2,[:V:rend:rendre:2]
       ([OBJ:N:livre:livre:4]))
7 (0.4,[:V:rend:rendre:2]
       ([A-OBJ:P:a:a:9]
          ([OBJ:N:biblio.:biblio.:11])))
8 (0.6,[:V:emprunte:emprunter:8]
       ([A-OBJ:P:a:a:9]
          ([OBJ:N:biblio.:biblio.:11])))
```

The actual IL program is the following:

- *Definition of the variables*

$\gamma_2^5, \gamma_2^6, \gamma_2^7$	verb *rendre* can be the root of ISC 5, 6, and 7
γ_8^8	verb *emprunter* can be the root of ISC 8
δ_1^5	noun *Jean* can be the leaf of ISC 5
δ_4^6	noun *livre* can be the leaf of ISC 6
$\delta_{11}^7, \delta_{11}^8$	noun *bibliothèque* can be the leaf of ISC 7 or 8

- *Definition of the constraints*
  - a word cannot be the leaf of more than one ISC
    - $\delta_1^5 \leq 1$ noun *Jean* can only be the leaf of ISC 5
    - $\delta_4^6 \leq 1$ noun *livre* can only be the leaf of ISC 6
    - $\delta_{11}^7 + \delta_{11}^8 \leq 1$ noun *bibliothèque* cannot be the leaf of both ISC 7 and 8
  - for an ISC to be selected, both its root and its leaf must be selected
    - $\gamma_2^5 - \delta_1^5 = 0$ ISC 5 must have a root and a leaf
    - $\gamma_2^6 - \delta_4^6 = 0$ ISC 6 must have a root and a leaf
    - $\gamma_2^7 - \delta_{11}^7 = 0$ ISC 7 must have a root and a leaf
    - $\gamma_8^8 - \delta_{11}^8 = 0$ ISC 8 must have a root and a leaf
- *Definition of the objective function*

$$\max(0.2\gamma_2^5 + 0.2\gamma_2^6 + 0.4\gamma_2^7 + 0.6\gamma_8^8)$$

The problem admits twelve solutions, represented in Figure 2. The optimal solution is solution number 12, in which the prepositional phrase *à la bibliothèque* is attached to the verb *emprunter*.

3.5 Combining Subcategorization Frames and Selectional Constraints

SF selection and SC satisfaction can be combined together in a single ILP program that combines the variables and constraints of the two problems and adds a new constraint that takes care of the incompatibilities of SCs and SFs.

Given a sentence S of length N, the set $\mathcal{I}$ of ISFs and the set $\mathcal{I}'$ of ISCs, the selection of the optimal set $\hat{\mathcal{I}}'' \subseteq \mathcal{I} \cup \mathcal{I}'$ is the solution of the following program:

- *Definition of the variables*
  $\alpha_i^j, \beta_i^j, \gamma_i^j, \delta_i^j$
- *Definition of the constraints*
  - All the constraints of ISF selection and ISC satisfaction.

	γ_2^5	δ_1^5	γ_2^6	δ_4^6	γ_2^7	δ_{11}^7	γ_8^8	δ_{11}^8	*score*
1	○	○	○	○	○	○	○	○	0.0
2	●	●	○	○	○	○	○	○	0.2
3	○	○	●	●	○	○	○	○	0.2
4	○	○	○	○	●	●	○	○	0.4
5	○	○	○	○	○	○	●	●	0.6
6	●	●	●	●	○	○	○	○	0.4
7	●	●	○	○	●	●	○	○	0.6
8	●	●	○	○	○	○	●	●	0.8
9	○	○	●	●	●	●	○	○	0.6
10	○	○	●	●	○	○	●	●	0.8
11	●	●	●	●	●	●	○	○	0.8
12	●	●	●	●	○	○	●	●	**1.0**

Figure 2
Representation of the 12 solutions to the ILP problem.

– Incompatible ISFs and ISCs cannot be selected together.
An ISF j and an ISC j' are not compatible if they share a common leaf (i') but have different roots ($i \neq i''$). Such a constraint can be modeled by the following inequality:

$$\forall i, i', i'', j, j' \; s.t. \; i \neq i'', \; 2\alpha_i^j + \beta_{i'}^j + 2\gamma_{i''}^{j'} + \delta_{i'}^{j'} \neq 5$$

- *Definition of the objective function*

$$\max \left(\sum_{j \in \mathcal{I}'} \delta_{R(j)}^j s_{SC}(j) + \sum_{j \in \mathcal{I}} \alpha_{R(j)}^j s_{SF}(j) \right)$$

The actual ILP program is too large and redundant with the two preceding ones to be exposed fully here. Its set of variables and constraints is the union of the variables and the constraints of the preceding ones, plus the two following ones:

$2\alpha_2^2 + \beta_{11}^2 + 2\gamma_8^8 + \delta_{11}^8 \neq 5$ ISF 2 and ISC 8 are incompatible
$2\alpha_8^4 + \beta_{11}^4 + 2\gamma_2^7 + \delta_{11}^7 \neq 5$ ISF 4 and ISC 7 are incompatible

Definition of the objective function:

$$\max(0.2\gamma_2^5 + 0.2\gamma_2^6 + 0.4\gamma_2^7 + 0.6\gamma_8^8 + 0.2\alpha_2^1 + 0.4\alpha_2^2 + 0.3\alpha_8^3 + 0.6\alpha_8^4)$$

The problem admits 94 solutions. The optimal solution is made of ISFs 1 and 4, and ISCs 5, 6, and 8.

4. Combining Parsing and Patching

Two processes have been described in Sections 2 and 3. In the first one, parsing is based on dynamic programming and produces a complete parse of a sentence S whereas in the second, patching is based on ILP and produces a partial parse of S: the set $\hat{\mathcal{I}}''$ made of ISCs and ISFs.

We would like now to produce a parse for sentence S that combines constraints coming from parsing and patching. We have exposed in Section 1 the reasons why we chose to keep these two processes separated and to combine their solutions to produce a single parse. The whole process is composed of three steps:

1. Sentence S is first parsed using a first-order parser and the set of k-best parses is produced in order to generate the set $\mathcal{I}$ of ISFs and the set $\mathcal{I}'$ of ISCs.
2. Patching is then performed and computes the optimal set $\hat{\mathcal{I}}'' \subseteq \mathcal{I} \cup \mathcal{I}'$ of ISFs and ISCs.
3. A new scoring function $s^+_{\hat{\mathcal{I}}''}$ is then computed and a second-order parser produces a parse tree $\hat{T}$ that preserves the ISFs and ISCs computed in Step 2.

This setting is strongly biased in favor of the patching process. Because of the nature of the scoring function $s^{+}_{\hat{\mathcal{I}}''}$, the ISFs and ISCs computed during the patching step are kept in the final solution even if considered unlikely by the parser. In practice, this solution does not perform well.

In order to let the parser influence the solution of the patching process, the confidence measure *CM*, defined in Section 2, is introduced in the objective function of the ILP. Recall that $CM(\Delta)$ indicates how confident the parser is about sub-tree Δ. This quantity is added to the scores of SFs and SCs introduced in sections 3.3 and 3.4.

The new scoring functions $\hat{s}_{SF}$ and $\hat{s}_{SC}$ are defined as follows:

$$\hat{s}_{SF}(\Delta) = (1 - \mu_1)s_{SF}(\Delta) + \mu_1 CM(\Delta)$$

$$\hat{s}_{SC}(\Delta) = (1 - \mu_2)s_{SC}(\Delta) + \mu_2 CM(\Delta)$$

where the values of μ_1 and μ_2 are determined experimentally on a development set. In the definition of $\hat{s}_{SF}$ and $\hat{s}_{SC}$, the first component (s_{SF} and s_{SC}) can be interpreted as a lexical score and the second one (*CM*) as a syntactic score.

After patching is performed and the set $\hat{\mathcal{I}}''$ is computed, the scoring function of the second order parser is modified in order to force the ISFs and ISCs of $\hat{\mathcal{I}}''$ to appear in the parser's solution.

As explained in Section 2.2, the modification of the scoring function modifies only the scores of first-order factors (single dependencies). The modification amounts to setting to $-\infty$ the scores of the dependencies that compete with those dependencies that occur in $\hat{\mathcal{I}}''$. At this point, the set $\hat{\mathcal{I}}''$ is just considered as a set of first-order factors, whatever the ILSC they are a member of. The scores of second-order factors are not modified. This solution works because the second-order parser uses both first- and second-order factors. There is no need to modify the scoring function of the second- (or higher) order factors since the first-order factors that they are composed of will be discarded anyway, because of their modified scores.

Let us illustrate the process on the sentence *Jean commande une glace à la serveuse.* [Jean orders an ice-cream from the waitress.]. The parser output for this sentence wrongly attaches the preposition *à* to the noun *glace* whereas it should be attached to the verb *commander*. Suppose that, after patching, the set $\hat{\mathcal{I}}''$ is made of the ISF (*commander, SBJ:N OBJ:N AOBJ:N*): the verb *commander* takes an indirect object introduced by preposition *à*, and the ISC (*VaN, commander, serveuse*): the noun *serveuse* is the indirect object of the verb *commander*. At this point $\hat{\mathcal{I}}''$ is viewed as the set of dependencies { (*commande, SUJ, Jean*), (*commande, OBJ, glace*), (*commande, AOBJ, à*), (*à, OBJ, serveuse*)}. The score of all competing dependencies, which are all dependencies that have either *Jean*, *glace*, *à*, or *serveuse* as a dependent, are set to $-\infty$ and will be discarded from the parser solution.

5. Complexity of the Patching Problem

The two-stage architecture proposed in this paper—parsing using dynamic programming; then patching using ILP—suffers from a potential efficiency problem because of the complexity of ILP, which is NP-complete (Karp 1972) in the general case. Our system has therefore an exponential worst-case complexity. Two questions then arise:

Could patching be solved using a polynomial algorithm? How does a general ILP solver perform, in practice, on our data?

We will answer these questions in two steps. First, we will prove in this section that patching is actually NP-complete. Building the optimal solution requires exponential time in the worst case. Second, we will show in Section 7, that, due to the size of the instances at hand, the processing time using a general ILP solver is reasonable.

In order to prove that patching is NP-complete, we will first show that some instances of patching can be expressed as a Set Packing problem (SP), known to be NP-complete (Karp 1972). The representation of a patching problem as a SP problem will be called patching as a Set Packing problem (P-SP). We will then reduce SP to P-SP by showing that any instance of SP can be represented as an instance of P-SP, which will prove the NP-completeness of P-SP and therefore the NP-completeness of patching.

The Set Packing decision problem is the following: Given a finite set $\mathcal{S}$ and a list $\mathcal{U}$ of subsets of $\mathcal{S}$, are there k subsets in $\mathcal{U}$ that are pairwise disjoint? The optimization version of SP, the Maximum Set Packing problem (MSP), looks for the maximum number of pairwise disjoint sets in $\mathcal{U}$.

We will show that a subset of the instances of patching can be expressed as an MSP. More precisely, we restrict ourselves to instances such that:

- the set $\mathcal{I}'$ of Instantiated Constraint Selections is empty (we only deal with a set $\mathcal{I}$ of ISFs) and
- the scores of all ISFs in $\mathcal{I}$ are equal.

The reason why we limit ourselves to a subset of patching is that our goal is to reduce any SP problem to a patching problem. Reducing an SP problem to some special cases of patching is enough to prove NP-completeness of patching. Patching in the general case might be represented as an MSP problem but this is not our goal.

Recall that patching with ISF is subject to three constraints:

C_1: a word cannot be the predicate of more than one ISF

C_2: a word cannot be the argument of more than one ISF

C_3: for an ISF to be selected, both its predicate and its arguments must be selected

Given a sentence $S = w_1, \ldots, w_n$ and a set $\mathcal{I}$ of ISFs on S, let us associate with S the set $\mathcal{S} = \{1, \ldots, n\}$. As a first approximation, let us represent an ISF of $\mathcal{I}$ as the subset of $\mathcal{S}$ composed of the indices of the arguments of the ISF. The list of ISF $\mathcal{I}$ is therefore represented as a list $\mathcal{U}$ of subsets of $\mathcal{S}$. Solving the MSP problem for $(\mathcal{S}, \mathcal{U})$ produces the optimal solution $\hat{\mathcal{U}}$ that satisfies C_2 and partly C_3. C_2 is satisfied because the elements of $\hat{\mathcal{U}}$ are pairwise disjoint. C_3 is partly satisfied because all the arguments of an ISF are selected altogether, but not the predicate.

The most straightforward way to take C_1 into account is to add the index of an ISF predicate in the subset corresponding to the ISF. Unfortunately, this solution is not correct. It will correctly constrain a word to be the predicate of at most one ISF but it will incorrectly prevent a word from acting as a predicate of an ISF and as an argument of another ISF. In order to solve this problem, we will associate two integers to a word of the sentence, one representing the word as a predicate and the other representing it as an argument. Given a sentence of length n and the word of index i, i will represent the word as an argument and $i + n$ will represent the word as a predicate. An ISF

having word w_p as a predicate and words $w_{a_1} \dots w_{a_k}$ as arguments is represented as the subset $\{a_1, \dots a_k, p+n\}$. The solution to the MSP problem using this representation clearly verifies our three constraints.

Given a sentence $S = w_1, \dots, w_n$ and a set of ISFs $\mathcal{I}$, we can build the set $\mathcal{S} = \{1, \dots, N\}$, with $N \geq 2n^4$ and the list $\mathcal{U}$ such that any element of $\mathcal{U}$ corresponds to an ISF of $\mathcal{I}$, using the method described here. The solution to the MSP on input $(\mathcal{S}, \mathcal{U})$ is equivalent to the solution of patching on input $(S, \mathcal{I})$. Such an MSP problem will be called a P-MSP problem (patching as Maximum Set Packing). Recall that the decision version of the problem is called P-SP (patching as Set Packing).

In order to prove the NP-completeness of patching, we will reduce any instance of SP to an instance of P-SP. Given the set $\mathcal{S} = \{1, \dots, n\}$ and the list $\mathcal{U} = (\mathcal{U}_1, \dots, \mathcal{U}_m)$ of subsets of $\mathcal{S}$, let us build the set $\mathcal{S}' = \{1, \dots, n+m\}$ and the list $\mathcal{U}' = (\mathcal{U}'_1, \dots, \mathcal{U}'_m)$ such that $\mathcal{U}'_i = \mathcal{U}_i \cup \{i+n\}$. The pair $(\mathcal{S}', \mathcal{U}')$ is a valid instance of P-SP. The solution of P-SP on input $(\mathcal{S}', \mathcal{U}')$ corresponds to the solution of SP on input $(\mathcal{S}, \mathcal{U})$.[5] If there was a polynomial algorithm to solve patching, there would be a polynomial algorithm to solve SP. SP would therefore not be NP-complete, which is incorrect. Patching is therefore NP-complete.

6. Experimental Set-up

We describe in this section and the two following ones the experimental part of this work. This section concerns language-independent aspects of the experimental set-up and Sections 7 and 8 describe the experiments and results obtained, respectively, on French and English data.

The complete experiments involve two offline and three online processes. The offline processes are:

1. Training the parser on a treebank.
2. Extracting SFs and SCs from raw data and computing their scores. The result of this process is the set $\mathcal{L}$.

The online processes are:

1. Generating the set $\mathcal{I}$ of ISFs and ISCs on a sentence S, using $\mathcal{L}$. We will call this process **candidate generation**.
2. Patching sentence S with $\mathcal{I}$, using ILP. The result of this process is the set $\hat{\mathcal{I}}$.
3. Parsing S under the constraints defined by $\hat{\mathcal{I}}$.

The remaining part of this section is organized as follows: Section 6.1 gives some details about the parser that we use; Section 6.2 describes some aspects of the extraction of SCs and SFs from raw data. Section 6.3 focuses on the candidate generation process.

4 At this point, we can impose that $N = 2n$. We will need the condition $N \geq 2n$ later in order to prove the reducibility of SP to P-SP. The condition $N \geq 2n$ is not harmful, if $N > 2n$, the range $[2n+1, \dots, N]$ will simply not be used when representing a patching instance as a P-SP instance.

5 It is important to note that in the set $\mathcal{U}'_i = \mathcal{U}_i \cup \{i+n\}$, adding the new element $i+n$ does not introduce any new constraint because this element only occurs in the set $\mathcal{U}'_i$ and will not conflict with any element of another set $\mathcal{U}'_k$ with $k \neq i$.

We have decided not to devote specific sections to the processes of patching and parsing under constraints. Patching has been described in full in Section 3. The ILP solver we have used is SCIP (Achterberg 2009). Parsing under constraints has been described in section 2.2.

6.1 Parser

The parser used for our experiments is the second-order graph-based parser implementation of Bohnet (2010). We have extended the decoding part of the parser in order to implement constrained parsing and *k*-best parses generation. The first-order extension is based on algorithm 3 of Huang and Chiang (2005), and the second-order extension relies on a non-optimal greedy search algorithm.

The parser uses the 101 feature templates, described in Table 4 of Bohnet (2010). Each feature template describes the governor and dependent of a labeled dependency, called the **target dependency**, as well as some features of its neighborhood. Every feature is associated with a score, computed on a treebank. A parse tree score is defined as the sum of the features it contains and the parser looks for the parse tree with the highest score.

Feature templates are divided into six sets:

Standard Feature Templates are the standard first-order feature templates. They describe the governor and the dependent of the target dependency. The governor and the dependent are described by combinations of their part of speech, lemma, morphological features, and fledged form.

Linear Feature Templates describe the immediate linear neighbors of the governor and/or the dependent of the target dependency. In such templates, governor, dependent, and neighbors are represented by their part of speech.

Grandchild Feature Templates are the first type of second-order features. They describe the structural neighborhood of the target dependency. This neighborhood is limited to a child of the dependent of the target dependency. Such templates are suited to model linguistic phenomena such as prepositional attachments.

Linear Grandchild Feature Templates describe the immediate linear neighbors of the governor, the dependent, and the grandchild of the target dependency.

Sibling Feature Templates are the second type of second-order feature templates. They describe the structural neighborhood of the target dependency limited to a sibling of the dependent of the target dependency. Such templates are suited, for example, to prevent the repetition of non-repeatable dependencies.

Linear Sibling Feature Templates describe the immediate linear neighbors of the governor, and the two siblings.

When run in first-order mode, the parser only uses Standard and Linear features. All features are used when the parser is run in second-order mode.

Four measures are used to evaluate the parser. The first two are standard whereas the third and fourth have been specifically defined to monitor the effect of our model on the performances of the parser:

Labeled Accuracy Score (LAS) of a tree T is the ratio of correct labeled dependencies in T.

Unlabeled Accuracy Score (UAS) of a tree T is the ratio of correct unlabeled dependencies in T.

Subcategorization Frame Accuracy Score (SFAS) of a tree T is the ratio of verbs in T that have been assigned their correct subcategorization frame. More precisely, given the reference parse R for sentence S and the output H of the parser for the same sentence, all ISFs are extracted from R and H, yielding the two sets R_{SF} and H_{SF}. SFAS is the recall of H_{SF} with respect to R_{SF} ($\frac{|R_{SF} \cap H_{SF}|}{|R_{SF}|}$).

Selectional Constraint Accuracy Score (SCAS) of a tree T is the ratio of correct occurrences of SC patterns in T. More precisely, given a gold standard tree R for sentence S and the output H of the parser for the same sentence, all ISCs are extracted from R and H, yielding the two sets R_{SC} and H_{SC}. SCAS is the recall of H_{SC} with respect to R_{SC}.

6.2 Extracting SFs and SCs from Raw Data

The LSC extraction process takes as input raw data and produces a set $\mathcal{L}$ composed of SCs and SFs. The process is straightforward: The corpora are first parsed, then LSC templates are applied on the parses, using unification, in order to produce SCs and SFs along with their number of occurrences. These numbers are used to compute selectional constraints scores (s_{SC}), as defined in Section 3.4, and subcategorization frame scores (s_{SF}) as described in Section 3.3.

The extraction of lexical co-occurrences from raw data has been a very active direction of research for many years. Giving an exhaustive description of this body of work is clearly beyond the aim of this article. The dominant approach for extracting lexical co-occurrences, as described, for example, in Volk (2001), Nakov and Hearst (2005), Pitler et al. (2010), and Zhou et al. (2011) directly model word co-occurrences on word strings. Co-occurrences of pairs of words are first collected on raw corpora or n-grams. Based on the counts produced, lexical affinity scores are computed. The detection of pairs of word co-occurrences is generally very simple: It is either based on the direct adjacency of the words in the string or their co-occurrence in a window of a few words. Bansal and Klein (2011) and Nakov and Hearst (2005) rely on the same sort of techniques but use more sophisticated patterns, based on simple paraphrase rules, for identifying co-occurrences. Our work departs from these approaches by extracting co-occurences in specific lexicosyntactic contexts that correspond to our SC patterns. Our approach allows us to extract more fine-grained phenomena but, being based on automatically parsed data, it is more error-prone.

Extracting SFs for verbs from raw data has also been an active direction of research for a long time, dating back at least to the work of Brent (1991) and Manning (1993). More recently, Messiant et al. (2008) proposed such a system for French verbs. The method we use for extracting SF is not novel with respect to such work. Our aim was not to devise new extraction techniques, but merely to evaluate the resource produced by such techniques for statistical parsing.

6.3 Candidate Generation

The candidate generation process has as input a sentence S and a set $\mathcal{L}$ of SCs and SFs. It produces the set $\mathcal{I}$ made of instantiations of elements of $\mathcal{L}$ on S. This set constitues the input of the ILP solver that will produce the solution $\hat{\mathcal{I}}$. The candidate generation

process is very important because it defines the search space of the patching process. If the search space is not wide enough, it might fail to contain the correct solution. If it is too wide, the IL programs generated will be too large to be solved in a reasonable amount of time.

Several methods can be used in order to perform the candidate generation process. One could compute $\mathcal{I}$ on the linear representation of S enriched with part of speech tags and lemmas. Bechet and Nasr (2009), for example, represent LSCs as finite-state automata that are matched on S. This method has a tendency to over-generate and proposes some very unlikely instantiations.

In order to constrain the instantiation process, some syntax can be used. For example, LSCs can be matched on a set $\mathcal{T}$ of possible parses of S in order to produce $\mathcal{I}$. The set $\mathcal{T}$ itself can be the k-best parses produced by a parser, as in parse reranking (Collins and Koo 2005; Charniak and Johnson 2005). It can also be built using parse correction techniques (Hall and Novák 2005; Attardi and Ciaramita 2007; Henestroza and Candito 2011). In the latter approach, a parse tree T of S is first built, using a parser, and parse correction rules are applied on it in order to build other parses that might correct errors of T.

The solution that we propose uses a parser to produce a list $\mathcal{T} = \{T_1, \ldots, T_k\}$ of k-best parses, such as in parse reranking. But this list is merged into a set of dependencies $\mathcal{D}_k$ that is the union of all the dependencies appearing in the k-best trees. The set $\mathcal{D}_k$ is not a tree because a single word can have several governors. This set can be seen as a superset of the k-best parses. We will call this method **sub-parse combination** or simply **combination** when the context is not ambiguous.

The main difference between reranking and combination is that the latter can combine dependencies that do not co-occur in a single tree of the k-best list. The search space that it defines is therefore larger than the k-best list. The set $\mathcal{D}_k$ can be compared to the pruned forest of Boullier, Nasr, and Sagot (2009). Once the set $\mathcal{D}_k$ is built, the LSCs of $\mathcal{L}$ are matched on the dependencies of $\mathcal{D}_k$ using unification.

In order to compare the search spaces respectively defined by a k-best parse list and the corresponding $\mathcal{D}_k$ set, we define two measures on the k-best parse list. The first one (Oracle LAS) corresponds to the LAS of the best tree in the k-best list. The second one (Recall LAS) is the part of labeled dependencies of the reference parse that appear in $\mathcal{D}_k$. Oracle LAS constitutes an upper bound for reranking and Recall LAS is an upper bound for combination. The values of these two measures are represented in Figure 3

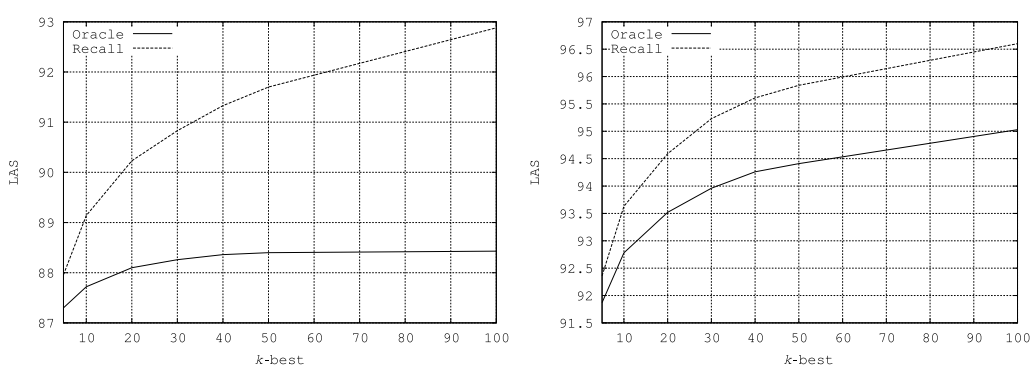

Figure 3
Oracle and Recall LAS of the k-best trees for first-order (left) and second-order (right) parsing, computed on FTB TEST.

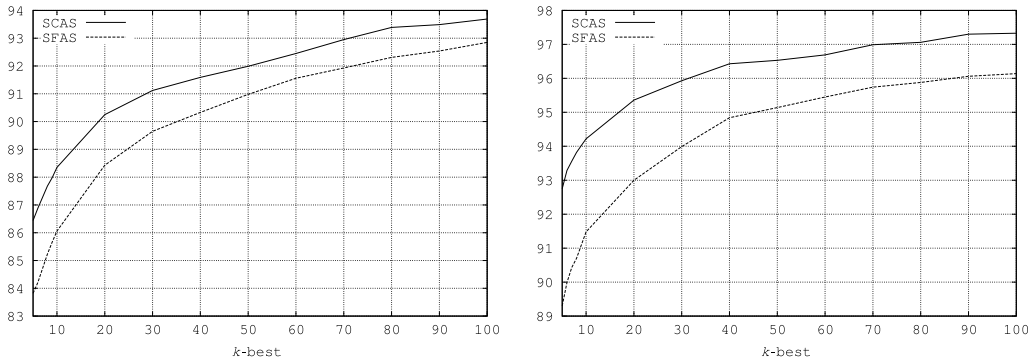

Figure 4
Recall SFAC and SCAS of the k-best trees for first-order (left) and second-order (right) parsing, computed on FTB TEST.

for first-order parsing (left) and second-order parsing (right) measured on the test set of the French Treebank.

As one can see in Figure 3, the Recall LAS curve is, of course, always above the Oracle LAS curve. In the case of first-order parsing, the Oracle curve quickly reaches a plateau. This is because first-order k-best parses are generally very close to one another, because of the very local nature of the model. The parses in the k-best list tend to be combinations of dependencies that appeared in previous parses. The number of new dependencies created tend to decrease quickly with k. For $k = 100$, the difference between the Oracle and the Recall scores is almost equal to 5 points. This difference clearly demonstrates the advantage of using $\mathcal{D}_k$ instead of the k-best list.

The results for second-order parsing are less spectacular. The Oracle curve does not reach an asymptote, as it was the case for first-order parsing. New dependencies are continually being created as k increases. The reason why we have considered first-order models in this section is that generating a first-order k-best solution can be done very efficiently; second-order k-best generation is more time-consuming. Performing candidate generation with a first-order parser is therefore a potentially interesting method. We will see in the two following sections that first-order k-best generation gave good results on French but did not perform well on English, where second-order k-best generation had to be used.

Although Figure 3 gives an estimation of the LAS upper bounds of reranking and combination, it does not tell us what can be expected in terms of SFAS and SCAS. Figure 4 answers this question by showing the evolution of Recall SFAS and Recall SCAS on the same data set. As one can see, for $k = 100$, with first-order k-best generation, almost 94% of the ISCs and 93% of the ISFs that appear in the reference can be recovered. The results for second-order k-best are, respectively, equal to 97% and 96%.

7. Experiments on French

We describe in this section the experiments performed on French and the results obtained. We start, in Section 7.1, with the description of the data used to train the parser and give an overview of the errors made by the parser. The extraction of SCs and SFs from raw corpora is described in Section 7.2, which starts with the description of the corpora that were used, then gives general statistics and coverage results for

Table 1
Decomposition of the French Treebank into training, development, and test sets.

	number of sentences	number of tokens
FTB TRAIN	9,881	278,083
FTB DEV	1,239	36,508
FTB TEST	1,235	36,340

the extracted data. The results of patching and parsing are given in Section 7.3 and an analysis of the errors is presented in Section 7.4. Section 7.5 analyzes the runtime performances.

7.1 Parsing

The parser was trained on the French Treebank (Abeillé, Clément, and Toussenel 2003), transformed into dependency trees by Candito et al. (2009). The size of the treebank and its decomposition into train, development, and test sets are represented in Table 1.

The parser gave state-of-the-art results for French: the LAS and UAS are, respectively, 88.88% and 90.71%. The SCAS reaches 87.81% and the SFAS is 80.84%, which means that in 80.84% of the cases, a verb selects its correct subcategorization frame. These results are for sentences with gold part-of-speech-tags and lemmas.

An analysis of the errors shows that the first cause of errors is prepositional attachments, which account for 23.67% of the errors. A more detailed description of the errors is given in Table 2. Every line of the table corresponds to one attachment type. The first column describes the nature of the attachment. The second indicates the frequency of this type of dependency in the corpus. The third column displays the accuracy for this type of dependency (the number of dependencies of this type correctly predicted by the parser divided by the total number of dependencies of this type). The fourth column shows the contribution of the errors made on this type of dependency to the global error rate. We have used this error analysis to define the following four SC patterns, which we call SBJ, OBJ, VdeN, and VaN:

```
SBJ  ([:V:::]([SUJ:N:::]))
OBJ  ([:V:::]([OBJ:N:::]))
VdeN ([:V:::]([DE-OBJ:P:de::]([OBJ:N:::])))
VaN  ([:V:::]([A-OBJ:P:a::]([OBJ:N:::])))
```

The last column of Table 2 indicates the SC pattern that corresponds to a given type of error. One can see that all SC patterns are not described at the same level of detail—some of them specify the label of the dependency, if considered important (to distinguish subject and object, for example), and others specify the lexical nature of the dependent (for prepositions, for example).

7.2 Extracting SFs and SCs

In order to automatically produce the set $\mathcal{L}$ of SCs and SFs, we gathered a raw corpus that we have parsed using the parser described in Section 7.1. The corpus is actually a collection of three corpora of slightly different genres. The first one (AFP) is a collection of news reports of the French press agency *Agence France Presse*; the second (EST REP) is a collection of newspaper articles from a local French newspaper: *l'Est Républicain*.

Table 2
The seven most common error types made by the baseline parser. For each type of error, we displayed its frequency, the parser accuracy, the impact on global errors, and the SC pattern that corresponds to the error type.

dependency	freq.	acc.	contrib.	SC pattern
N—mod →N	1.50	72.23	2.91	
V—∗ → à	0.88	69.11	2.53	VaN
V—suj → N	3.43	93.03	2.53	SBJ
N—coord → CC	0.77	69.78	2.05	
N—∗ → de	3.70	92.07	2.05	
V—∗ → de	0.66	74.68	1.62	VdeN
V—obj → N	2.74	90.43	1.60	OBJ

The third (WIKI) is a collection of articles from the French Wikipedia. The size of the different corpora are detailed in Table 3.

The raw corpora were first tokenized, POS-tagged, and lemmatized with the MACAON tool suite (Nasr et al. 2011), then parsed in order to get the most likely parse for every sentence. The SFs and SCs were then extracted from the parses and their scores computed. In the two following sections, we give some statistics on the SFs and SCs produced.

7.2.1 Extracted SF. As described in Section 3.3, an SF frame is a special kind of LSC. Its root is a predicate and its leaves are the arguments of the predicate. Some linguistic constraints are defined over SF. The first one is the category of the root, which must be either a finite tense verb (V), an infinitive form (VINF), a past participle (VPP), or a present participle (VPR). The second is the set of syntactic functions that introduce arguments of the predicate; they can either be a subject (SUJ), a direct object (OBJ), or an indirect object introduced by preposition *à* (A-OBJ) or *de* (DE-OBJ).

SFs are defined at a level that is slightly more abstract than the surface syntactic realization. Two abstractions are in action. The first one is the factoring of several parts of speech. We have factored pronouns, common nouns, and proper nouns into a single category N. We have not gathered verbs of different modes into one category because the mode of the verb influences its syntactic behavior and hence its SF. The second abstraction is the absence of linear order between the arguments.

The number of verbal lemmas, number of SFs, and average number of SFs per verb are shown in Table 4, column A_0. As can be observed, the number of verbal lemmas and

Table 3
Some statistics computed on the corpora used to collect Subcategorization Frames and Selectional Constraints.

CORPUS	Number of sentences	Number of tokens
AFP	2,041,146	59,914,238
EST REP	2,998,261	53,913,288
WIKI	1,592,035	33,821,460
TOTAL	5,198,642	147,648,986

Table 4
Number of verbal lemmas, SFs, and average number of SFs per verb for three levels of thresholding (0, 5, and 10).

	A_0	A_5	A_{10}
number of verbal lemmas	23,915	4,871	3,923
number of different SFs	12,122	2,064	1,355
average number of SFs per verb	14.26	16.16	13.45

SFs are unrealistic. This is because the data from which SFs were extracted are noisy: they consist of automatically produced syntactic trees that contain errors. There are two main sources of errors in the parsed data: the pre-processing chain (tokenization, part-of-speech tagging, and lemmatization), which can consider as a verb a word that is not; and, of course, parsing errors, which tend to create incorrect SFs. In order to fight against noise, we have used a simple thresholding: We only collect SFs that occur more than a threshold i. The results of the thresholding appear in columns A_5 and A_{10}, where the subscript is the value of the threshold. It is a very crude technique that allows us to eliminate many errors but also rare phenomena. As expected, both the number of verbs and SFs decrease sharply when increasing the value of the threshold. The average number of SFs per verb is 14.26 without threshold and reaches 16.16 for a threshold of 5. This phenomenon is mainly due to part of speech tagging errors on the raw corpus: A large part of hapax are words mistakenly tagged as verbs. They disappear with thresholding, and their disappearance tends to increase ambiguity.

We report in Table 5 the coverage of the extracted SFs on FTB DEV. The last column represents the coverage of the SF extracted from FTB TRAIN. These figures constitute an interesting reference point when evaluating the coverage of the SFs extracted from raw data. We have distinguished lexical coverage (ratio of verbal lemmas of FTB DEV present in the resource) and syntactic coverage (ratio of SFs of FTB DEV present in the resource). Both measures were computed on types and tokens. Table 5 shows that lexical coverage is very high, even on A_{10}. It is interesting to note that syntactic coverage of the automatically built resource is much higher than syntactic coverage of the same resource extracted from FTB TRAIN. This clearly shows that many SFs of FTB DEV were not seen in FTB TRAIN. This observation supports our hypothesis that treebanks are not large enough to accurately model the possible SFs of a verb. Comparing coverage of T and A_x is somehow unfair because, as we already mentioned, the automatic resources are noisy, and have a tendency to overgenerate whereas T contains only correct SFs (up to the corpus annotation errors). Accuracy results of Section 7.3 will give a more balanced

Table 5
Lexical and syntactic coverage on FTB DEV of the extracted SFs. Coverage is computed for three thresholding levels of the automatically generated resource (A_0, A_5, and A_{10}) and FTB TRAIN (T).

	A_0		A_5		A_{10}		T	
	types	tokens	types	tokens	types	tokens	types	tokens
Lex. cov.	99.52	99.85	98.56	99.62	98.08	99.50	89.56	96.98
Synt. cov.	95.78	97.13	91.08	93.96	88.84	92.39	62.24	73.54

Table 6
Number of occurrences of the four SC patterns (OBJ, SBJ, VdeN, and VaN) in raw corpora with three levels of thresholding (0, 5, and 10).

SC pat.	A_0	A_5	A_{10}
OBJ	422,756	58,495	26,847
SBJ	433,196	55,768	25,291
VdeN	116,519	11,676	4,779
VaN	185,127	23,674	10,729
TOTAL	1,157,598	149,613	67,646

view by computing how many errors are actually corrected using the noisy extracted resource.

7.2.2 Extracted SCs. The number of different SCs extracted for each of the four SC configurations are shown in Table 6 for the three thresholds 0, 5, and 10.

The coverage of SCs on FTB DEV is shown in Table 7. It is interesting to note that SC coverage is much lower than SF coverage. This is due to the different nature of the two phenomena: SC is a bilexical phenomena whereas SF is a monolexical one. Ideally, a larger corpus should be used to extract SCs. One can also note that the coverage of SCs extracted from FTB DEV is very low.

7.3 Results

The results obtained by our method are shown in Table 8, using the four measures defined in Section 6.1. The accuracy scores of the baseline parser are presented in line 1. Line 2 shows the results obtained when setting parameters μ_1 and μ_2 to 1. This setting corresponds to the situation where SF and SC scores are not used. Patching, in this case, only combines ILSCs that have a good confidence measure. This experiment is important in order to ensure that s_{SF} and s_{SC} do carry useful information. As one can see, setting μ_1 and μ_2 to 1 slightly increases the accuracy of the parser, which means that the accuracy of the parser can be increased just by imposing the final parse sub-parses that have a good confidence measure.

The optimal values of μ_1 and μ_2 were computed on FTB DEV; they are both 0.65. These values are used for the experiments corresponding to lines 3, 4, and 5. Lines 3

Table 7
Coverage for the four SC configurations (OBJ, SBJ, VdeN, VaN). Coverage is computed for three thresholding levels of the automatically generated resource (A_0, A_5, and A_{10}) and FTB TRAIN (T).

SC pat.	A_0		A_5		A_{10}		T	
	types	tokens	types	tokens	types	tokens	types	tokens
OBJ	68.31	69.71	58.50	61.51	53.24	56.12	17.94	21.31
SBJ	66.26	68.18	52.87	57.24	47.82	52.94	23.11	24.48
VdeN	46.28	56.61	34.93	49.57	31.66	46.73	11.57	15.77
VaN	62.93	64.69	48.89	52.42	42.68	47.56	20.06	23.34
TOTAL	65.17	67.54	53.10	57.15	47.92	52.78	19.96	22.24

Table 8
Four measures of accuracy on FTB TEST for four settings: Baseline second-order parser, patching with confidence measure, patching with SF only, patching with SCs only, and patching with SFs and SCs.

	LAS	UAS	SCAS	SFAS
Baseline	88.88	90.71	87.81	80.84
CM only	89.02	90.98	88.03	81.16
SF Selection	89.21	91.05	88.19	84.13
SC Satisf.	89.22	91.09	91.97	82.03
Combined	**89.51**	**91.39**	**92.89**	**85.06**

and 4 indicate the accuracy for SF selection and SC satisfaction alone, as described in Sections 3.3 and 3.4. Line 5 provides the result when they are combined, as described in Section 3.5.

The table shows that, when setting μ_1 and μ_2 to their optimal values, the three methods increase both LAS and UAS with respect to the baseline. The best results are obtained by the combined method. SCAS and SFAS allow us to gain a better understanding of what is happening. As could be expected, SF selection has a stronger impact on SFAS whereas SC satisfaction has a stronger impact on SCAS. But both methods also have a positive influence on the other measure. This is because SFs and SCs are not independent phenomena and correcting one can also lead to correcting the other. The combined method yields the best results on our four measures.

7.4 Error Analysis

There are three main causes for SC and SF errors: insufficient coverage of $\mathcal{L}$, narrowness of the patching search space $\mathcal{I}$, or inadequacy of the ILP objective function. Given a sentence S, let us note R its correct parse and H the parse produced by our system. Let us note $\hat{\mathcal{I}}_R$ and $\hat{\mathcal{I}}_H$, respectively, the set of ILSCs of R and H. We define an error as an ILSC $x \in \hat{\mathcal{I}}_R$, such that $x \notin \hat{\mathcal{I}}_H$, and we note y the LSC corresponding to x (x is an instantiation of y). The three error causes are described as follows:

NotInDB $y \notin \mathcal{L}$. This is the situation where an ILSC cannot be extracted from the k-best list because its corresponding LSC does not appear in $\mathcal{L}$. A possible cause for such errors comes from the corpora we have used to extract LSCs. We have shown in Section 7.2 that the coverage of our resources is not perfect and some ILSCs that appear in the reference parse R might not appear in the resource. A solution to such a problem would be to use larger corpora in order to increase coverage. Another cause of this error is the processing chain that was used to process the raw corpora. As already mentioned, any of the modules that compose this chain can make errors and introduce noise in $\mathcal{L}$.

NotInKB $x \notin \mathcal{I} \cup \mathcal{I}'$. This is the situation where x does not appear in any of the k-best trees. The problem is the limitation of the search space of the ILP. The solution to this problem is either to increase the size of the search space or to use another method to produce it, as proposed in Section 3.2. The simplest way to increase the search space size is to increase the length of the k-best list. However, increasing

Table 9
Raw number of errors, error decrease, and error distribution for each SC pattern, average of SC patterns, and SFs.

Type	err.	decr.	NotInDB	NotInKB	NotInSOL
OBJ	72	37.5%	0.55	0.44	0.22
SBJ	101	47.5%	0.41	0.30	0.47
VdeN	59	27.1%	0.42	0.70	0.21
VaN	132	45.4%	0.24	0.33	0.59
SC	364	41.6%	0.38	0.42	0.41
SF	807	22.0%	0.22	0.24	0.54

the search space yields larger IL programs and therefore the resolution time of the ILP.

NotInSOL $x \in \mathcal{I} \cup \mathcal{I}'$ but $x \notin \hat{\mathcal{I}}_R$. This is the situation where x is in the input of the ILP program but is not part of the solution. There are two possible causes for this situation. The first one comes from the scores s_{SF} and s_{SC} computed on the corpora. A correct ILSC might have been assigned a lower score than an incorrect one. This situation, in turn, can be due to different attachment preferences in the test and the raw corpora used to produce $\mathcal{L}$. The second cause is the confidence measure used to introduce a syntactic dimension to the ILP objective function, as described in Section 4. In some cases, an ILSC might have a good lexical score but a bad syntactic score, in which case it is ruled out.

Table 9 details the distribution of the errors over these three categories. Column 1 describes the nature of the attachment. The four first rows correspond to specific SC patterns, row 5 is the average of all SCs, and row 6 concerns SF selection. Column 2 indicates the number of errors made by the baseline parser. Column 3 indicates the decrease of the number of errors and columns 4, 5, and 6 give the distribution of the errors over the three error types defined earlier.

As one can see from Table 9, our method decreases by 41.6% the number of SC errors and by 22% the number of SF errors. The difference between these two figures comes from the fact that SFs are more complex phenomena that SC patterns. SFs can be composed of a large number of dependencies and an error in one of them leads to an error in the SF selection.

The SC errors are almost equally distributed along the three error types: In 38% of the cases, the correct SC is not $\mathcal{L}$; in 42% of the cases, the correct SC does not appear in the 100 best parses; and in 41% of the cases, the ILP fails to find the correct SC, although it is in the search space. There are three possible causes for the latter type of error: Recall that the objective function of the ILP is based on the scores of ILSCs, which are themselves a combination of a lexical and a syntactic score: $s(\Delta) = (1 - \mu)s_L(\Delta) + \mu s_S(\Delta)$. The cause of the error could be the lexical score (s_L), the syntactic score (s_S), or the weight (μ).

Most SF errors pertain to category NotInSOL: In 54% of the cases, the correct SF is in the input of the ILP but it is not part of the solution. This might be due to the definition of the s_{SF} score, which is a simple conditional probability.

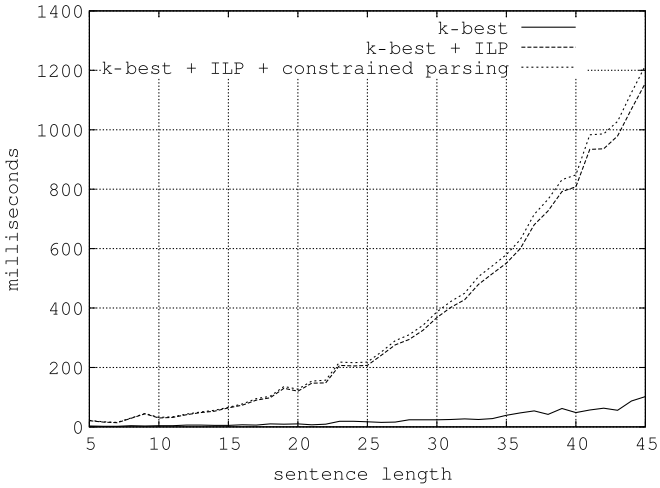

Figure 5
Decomposition of runtime on FTB TEST.

7.5 Runtime Analysis

Figure 5 displays the runtime of our system with respect to the length of the sentences parsed. Runtime has been decomposed in three parts: first-order 100-best generation, ILP solving, and constrained second-order parsing. The runtime of the instantiation process is not reported because it is negligible compared with the runtime of the other processes.

Figure 5 shows that ILP resolution is the most time-consuming part of the whole process. Although in practice the performances of the whole system are reasonable (approximately 330 words per second on average), we show that the patching problem is NP-complete and we have no guarantee that the problem will be solved in polynomial time.

8. Experiments on English

A second series of experiments was performed on English. Although the setting used is almost the same as for French, there are some notable differences between the two series of experiments with respect to the data used:

- The treebank used for training the parser is significantly larger.
- The definition of SC and SF patterns is different.
- The raw corpus used to extract ISCs and ISFs is one order of magnitude larger.

8.1 Parsing

The parser was trained and evaluated on the Penn Treebank (Marcus, Marcinkiewicz, and Santorini 1993). We have used the standard decomposition of the Penn Treebank

Table 10
Decomposition of the Penn Treebank into training, development, and test sets.

		Number of sentences	Number of tokens
PTB TRAIN	(Sec. 02–21)	39,279	958,167
PTB DEV	(Sec. 22)	1,334	33,368
PTB TEST	(Sec. 23)	2,398	57,676

into training, development, and test sets, with sizes as reported in Table 10. The main difference with respect to French comes from the training set, which is 2.44 times larger.

As could be expected, the parser trained and evaluated on the Penn Treebank yields better results than the same parser trained on the French Treebank. The LAS is 91.76 and UAS is 93.75, a relative increase of 3.24% and 3.35% with respect to French.

The English SF and SC definitions differ from the French ones in the choice of the preposition used. The nine most frequent prepositions occuring in the Penn Treebank were taken into account ({*of, in, to, for, with, on, at, from, by*}) to define both SCs and SFs. A set of 20 SC patterns are defined: {OBJ, SBJ, VofN, VinN, VtoN, VforN, VwithN, VonN, VatN, VfromN, VbyN, NofN, NinN, NtoN, NforN, NwithN, NonN, NatN, NfromN, NbyN}. SFs are defined as the dependents of a verb that are introduced by one of the nine prepositions, plus the subject and the object.

The SCAS and SFAS are, respectively, 91.99 and 88.24: an increase of 4.76% and 9.15% compared to French. These results show that the English parser makes fewer errors on prepositional attachments. This situation may be due to the fact that prepositional attachments are more ambiguous in French (4.56 prepositions on average per sentence in the FTB compared with 3.15 for the PTB).

Table 11 details the error rate for each of the 20 SC patterns defined, sorted with respect to their contribution to the total error rate. The total contribution of these errors reaches 18.87%, which is an approximation of the upper limit of the improvement that we can expect from our method. The mean accuracy for these patterns is almost equal to 92%. Two situations should be distinguished, however: the subject and direct object dependencies, on the one hand, and the prepositional attachments, on the other. The

Table 11
Contribution of the 20 SC patterns to the errors made by the parser. For each pattern, we displayed its frequency, the parser accuracy, and the impact on global errors.

SC pat.	freq.	acc.	contrib.	SC pat.	freq.	acc.	contrib.
VinN	0.88	72.75	2.91	VatN	0.20	78.26	0.53
OBJ	5.48	95.72	2.85	NofN	1.72	97.44	0.53
SBJ	7.71	97.17	2.65	VwithN	0.23	84.38	0.43
VforN	0.32	56.04	1.71	VbyN	0.34	92.82	0.30
NinN	0.59	76.35	1.69	NtoN	0.16	85.71	0.28
VonN	0.32	63.69	1.39	NwithN	0.15	84.52	0.28
VtoN	0.30	78.82	0.77	VofN	0.06	74.29	0.19
NforN	0.35	84.42	0.66	NatN	0.14	89.02	0.19
NonN	0.16	68.48	0.62	NfromN	0.10	85.96	0.17
VfromN	0.14	69.51	0.53	NbyN	0.06	76.47	0.17
				TOTAL	19.43	91.99	18.87

prepositional attachments (restricted to the nine most frequent prepositions) represent 6.24% of the dependencies of the test set but their contribution to the total error rate is equal to 13.37% and the mean accuracy of these attachments is equal to 82.32%, whereas subject and direct object dependencies correspond to 13.19% of the dependencies but generate only 5.5% of the errors.

8.2 Extracting SFs and SCs

The ISFs and ISCs were extracted from the Google-parsed Gigaword (Napoles, Gormley, and Van Durme 2012b, 2012a) (noted as the GIGA corpus in the remainder of the article), which is a subset of the original English Gigaword, fifth edition (Parker et al. 2011), parsed with the parser described in Huang, Harper, and Petrov (2010). The origin of the corpora as well as their sizes are reported in Table 12. As one can see, the size of the corpus is one order of magnitude larger than the French corpus we gathered to extract ISFs and ISCs—more precisely, it is 31 times larger.

The number of occurrences of each of the 20 SC patterns extracted are displayed in Table 13. A total of 33.6 million occurrences of SC patterns have been extracted.

Table 14 shows some statistics on the number of SF extracted from the data. The number of different SFs extracted is, on average, three times larger than in French. This difference is mainly due to the definition of English SFs, which take into account more prepositions. The mean number of SFs per verb is 15.39 without thresholding and reaches 17.73 for a threshold of 10.

Tables 15 and 16 display coverage on Penn Treebank (PTB) DEV for SC and SF extracted from the GIGA corpus, with three levels of thresholding, as well as the coverage for SCs and SFs extracted from PTB TRAIN.

The coverage for SCs is 65.66% on tokens, for a threshold equal to 10, a relative increase of 24% compared with French. This increase is the expected result of using more data. The most striking figure of Table 15 is the coverage of the training corpus. It was 22.24% for French and 54.55% for English. This increase cannot be explained only by the larger size of the training corpus for English. It seems to indicate a higher lexicosyntactic variety in the French Treebank than in the Penn Treebank, or a higher variety between the development and training sets of the French Treebank compared with the Penn Treebank.

Table 12
Some statistics computed on the corpora used to collect Subcategorization Frames and Selectional Constraints.

CORPUS	Number of sentences	Number of tokens
Agence France-Presse, English Service	35,200,934	811,564,028
Associated Press Worldstream, English Service	64,893,993	1,346,062,109
Central News Agency of Taiwan, English Service	1,399,398	41,125,514
Los Angeles Times/Washington Post Newswire Service	13,711,844	313,874,339
New York Times Newswire Service	80,630,849	1,672,223,270
Washington Post/Bloomberg Newswire Service	808,266	20,135,054
Xinhua News Agency, English Service	15,770,726	382,529,760
TOTAL	212,416,010	4,587,514,074

Table 13
Number of occurrences of the 20 SC patterns in the GIGA corpus with three levels of thresholding (0, 5, and 10).

SC pat.	A_0	A_5	A_{10}
OBJ	5,517,917	1,531,695	933,764
SBJ	5,150,122	1,312,531	768,133
VofN	488,475	79,799	37,128
VinN	1,866,766	471,104	270,713
VtoN	1,201,947	2,750,660	153,676
VforN	1,223,887	263,518	142,346
VwithN	1,591,669	302,467	151,949
VonN	1,148,819	260,671	142,715
VatN	640,019	143,864	80,390
VfromN	816,035	164,210	85,407
VbyN	1,200,835	252,166	131,468
NofN	4,655,268	1,048,782	579,842
NinN	2,028,525	404,431	212,265
NtoN	784,074	128,504	63,935
NforN	1,691,557	296,692	144,884
NwithN	1,066,320	138,280	62,051
NonN	956,125	170,875	85,611
NatN	516,497	93,686	47,476
NfromN	743,002	114,957	54,695
NbyN	347,791	56,020	27,363
TOTAL	33,635,650	7,509,318	4,175,811

Table 14
Number of verbal lemmas, SFs, and average number of SFs per verb for three levels of thresholding (0, 5, and 10).

	A_0	A_5	A_{10}
number of verbal lemmas	67,186	9,271	6,305
number of different SFs	31,055	6,394	3,816
average number of SFs per verb	15.39	21.52	17.73

The syntactic coverage of the SF extracted from the GIGA corpus (the ratio of pairs [verbal lemma, SF] of PTB DEV present in the GIGA corpus) is 77.56% when no threshold is applied, although it was 97.13% for French. This difference is the result of the more extended definition of SF in English. But the more surprising figure is the coverage of the PTB TRAIN, which is 82.7% (it was 73.54% for French). This difference could be explained, as was the case for SC coverage, by a difference of lexicosyntactic variety between the French Treebank and the Penn Treebank.

8.3 Results

In our experiments on French, the first step used a first-order *k*-best parser. In the English experiment, as already noted, the accuracy of the second-order parser is higher and the quality of the *k*-best parses using a first-order parser is not good enough to leave enough room for improvement over a second-order parser. The situation is detailed in

Table 15
Coverage for the 20 SC patterns on PTB DEV. Coverage is computed for three thresholding levels of the automatically generated resource (A_0, A_5, and A_{10}) and PTB TRAIN (T).

SC pat.	A_0		A_5		A_{10}		T	
SC pat.	types	tokens	types	tokens	types	tokens	types	tokens
OBJ	78.26	71.86	73.24	66.13	70.23	62.07	49.86	58.43
SBJ	79.18	82.66	75.99	78.25	74.03	76.34	68.83	75.97
VofN	86.36	86.96	54.55	56.52	50	52.17	13.64	17.39
VinN	72.86	73.94	67.29	68.66	65.8	67.25	28.62	32.04
VtoN	73.94	74.67	64.08	64.67	59.15	60	26.76	28.67
VforN	66.96	67.2	63.48	64.00	58.26	59.2	26.96	31.20
VwithN	82.35	83.02	72.55	73.58	70.59	71.7	11.76	13.21
VonN	79.55	78.26	73.86	72.83	69.32	68.48	27.27	30.43
VatN	86.05	87.23	83.72	85.11	79.07	80.85	27.91	29.79
VfromN	68.09	68.75	59.57	60.42	55.32	56.25	17.02	16.67
VbyN	59.05	59.63	51.43	52.29	45.71	45.87	9.52	11.01
NofN	68.54	68.07	60.94	60.53	56.42	56.14	27.49	28.77
NinN	58.14	59.12	52.33	53.59	48.26	49.17	26.16	28.73
NtoN	74.51	76.79	54.9	58.93	52.94	57.14	15.69	19.64
NforN	63.56	64.17	53.39	54.17	46.61	47.5	18.64	19.17
NwithN	46.34	47.62	43.9	45.24	36.59	38.1	9.76	11.90
NonN	67.27	66.07	52.73	51.79	45.45	44.64	27.27	28.57
NatN	50.00	51.22	47.5	48.78	37.5	39.02	22.5	21.95
NfromN	48.72	44.19	43.59	39.53	35.9	32.56	17.95	25.58
NbyN	52.17	52.17	39.13	39.13	34.78	34.78	0.00	0.00
TOTAL	74.59	74.8	69.17	69.01	65.95	65.66	46.6	54.55

Table 16
Lexical and syntactic coverage of the extracted SF on PTB DEV. Coverage is computed for three thresholding levels of the automatically generated resource (A_0, A_5, and A_{10}) and PTB TRAIN (T).

	A_0		A_5		A_{10}		T	
	types	tokens	types	tokens	types	tokens	types	tokens
Lex. cov.	98.34	99.37	98.23	99.35	95.58	98.77	95.36	98.95
Synt. cov.	72.60	77.56	66.23	72.87	63.16	70.97	70.00	82.70

Table 17. The table shows that the Oracle accuracy of 100 best parses for a first-order parser is often below the accuracy of the first-best parse for a second-order parser. The situation is slightly more favorable for $k = 200$ but is still dominated for some SCs by the first-best second-order accuracy. This is the reason why we decided to use a second-order parser in the candidate generation step. The rest of the process is unchanged.

The results obtained are shown in Table 18. The decrease of labeled error attachments is 3.15% and the decrease of unlabeled error attachments is 4.16%. These results are lower than what we observed for French, where the decrease is, respectively, 5.66% and 7.32%. Selectional Constraint violations are decreased by 16.21% and erroneous Subcategorization Frame assignments by 8.83%. They were equal to 41.6% and 22%, respectively, for French. These results are disappointing—we expected that the larger

Table 17
Unlabeled accuracy for each of the 20 SC patterns using a second-order and first-order parser. In the second-order parser, the first-best and oracle scores for $k = 100$ are shown. In the first-order parser, the first-best and the oracle score for $k = 100$ and $k = 200$ are shown.

	$o = 2$		$o = 1$		
SC pat.	100 oracle	first-best	100 oracle	200 oracle	first-best
OBJ	98.13	95.72	96.23	96.91	93.3
SBJ	98.93	97.17	96.39	96.71	93.96
VofN	100	74.29	85.71	88.57	77.14
VinN	96.59	72.75	61.32	63.73	56.91
VtoN	98.24	78.82	76.47	76.47	74.12
VforN	93.41	56.04	54.95	60.99	46.7
VwithN	99.22	84.38	85.16	87.5	82.03
VonN	94.41	63.69	60.34	60.89	56.42
VatN	98.26	78.26	57.39	59.13	53.91
VfromN	98.78	69.51	71.95	73.17	69.51
VbyN	98.97	92.82	89.23	89.74	88.21
NofN	99.49	97.44	98.57	98.88	97.03
NinN	96.11	76.35	81.14	82.34	70.66
NtoN	98.90	85.71	94.51	95.6	81.32
NforN	97.49	84.42	91.96	93.47	82.41
NwithN	96.43	84.52	90.48	94.05	77.38
NonN	95.65	68.48	72.83	73.91	64.13
NatN	100	89.02	90.24	91.46	85.37
NfromN	96.49	85.96	91.23	91.23	75.44
NbyN	100	76.47	91.18	97.06	82.35

Table 18
Four measures of accuracy on PTB TEST for four settings: Baseline second-order parser, patching with syntactic score only, patching with SFs only, patching with SCs only, and patching with SFs and SCs.

	LAS	UAS	SCAS	SFAS
Baseline	91.76	93.75	91.99	88.24
CM only	91.80	93.79	92.06	88.27
SF Selection	91.97	93.93	92.96	89.20
SC Satisf.	92.02	94.01	93.11	89.14
Combined	**92.05**	**94.05**	**93.29**	**89.28**

size of the corpus used for extracting ISCs and ISFs would yield a larger decrease of SC and SF errors.

8.4 Error Analysis

Table 19 allows us to gain a better understanding of the causes of the errors. The first cause of errors is the ILP objective function. In 53% of the cases for SC errors and 59% of the cases for SF errors, the ILP failed to find the correct SFs or SCs although it is in the search space. The inadequacy of the objective function is difficult to interpret because it has, itself, three causes, as we mentioned in our experiments on French: the SC or SF scores, the confidence measure, or the weight μ. The second cause of errors

Table 19
Raw number of errors, error decrease, and error distribution for each SC pattern, average of SC patterns, and SF.

Type	err.	decr.	NotInDB	NotInKB	NotInSOL
OBJ	133	14.29%	0.39	0.51	0.40
SBJ	124	4.03%	0.24	0.39	0.51
VofN	9	11.11%	0.38	0.00	0.63
VinN	136	30.15%	0.48	0.18	0.46
VtoN	36	30.56%	0.32	0.12	0.60
VforN	80	26.25%	0.29	0.2	0.61
VwithN	20	10.00%	0.44	0.06	0.56
VonN	65	18.46%	0.32	0.19	0.62
VatN	25	16.00%	0.19	0.10	0.71
VfromN	25	8.00%	0.35	0.04	0.65
VbyN	14	−14.29%	0.56	0.13	0.44
NofN	25	12.00%	0.23	0.23	0.68
NinN	79	17.72%	0.4	0.2	0.49
NtoN	13	0.00%	0.38	0.08	0.54
NforN	31	6.45%	0.24	0.17	0.59
NwithN	13	0.00%	0.31	0.23	0.46
NonN	29	24.14%	0.23	0.18	0.59
NatN	9	0.00%	0.44	0.00	0.56
NfromN	8	0.00%	0.13	0.25	0.75
NbyN	8	12.50%	0.43	0.00	0.57
SC	882	16.21%	0.34	0.25	0.53
SF	1314	8.83%	0.41	0.19	0.59

is coverage. Despite the size of the raw corpus used to extract SCs and SFs, the lack of coverage remains an important source of errors. The quality of the search space of the ILP is the third cause of errors. This observation shows that candidate generation with second-order parsing yields better results than first-order parsing.

9. Conclusions and Discussion

We showed in this paper that erroneous Subcategorization Frame assignment and Selectional Constraint violations can be partially corrected by using weighted lexicosyntactic patterns, called patches, representing Subcategorization Frames and Selectional Constraints that have been extracted from a large raw corpus. The experiments, performed on the French Treebank, allowed us to decrease by 41.6% Selectional Constraint violations and by 22% erroneous Subcategorization Frame assignments. These figures are lower for English: 16.21% in the first case and 8.83% in the second.

The method proposed is based on the decomposition of the parsing process into two sub-processes. The first one, called patching, identifies, in a sentence, a optimal set of patches that can be of arbitrary size. The second process is based on constrained graph-based parsing; it takes as input an optimal set of patches and builds a complete parse of the sentence that contains these patches.

The proposed architecture is one solution to the general problem of combining local and global lexicosyntactic patterns in a parser. It has been applied to the problem of selecting Subcategorization Frames and Selectional Constraints but it could be used to take into account any type of lexicosyntactic linguistic constraints, such as semantic frames or discourse structures.

The main limit of our method is the NP-complete nature of patching. In theory, solving an instance of the patching problem could require exponential time. In practice, for our task, the resolution time is reasonable, but it could become an issue when dealing with larger patches. Other approximate methods could be used to perform the patching task, this is the first direction in which we wish to continue this work.

The second direction that we want to explore concerns the first step of the patching process, namely, candidate generation. Patches being lexicosyntactic structures, the generation process needs to have access to the possible syntactic structures of the sentences. The solution we have proposed uses a first- or second-order *k*-best list of parses from which syntactic material will be used to propose possible patches. Another solution would be to perform the patch generation directly on the parse forest.

The error analysis showed that coverage of the automatically extracted ISCs and ISFs is an important cause of errors, despite the size of the corpus used to extract ISCs and ISFs. Collecting larger corpora might not be the right solution to increase coverage. The third direction we would like to explore is generalizing over the events that are observed in the data, using distributional semantics.

Acknowledgments

The authors thank Yann Vaxes for his help in proving the NP-Completeness of the patching problem and Fiona Torbey, Jean Torbey, and Balamurali Andiyakkal Rajendran for proofreading.

References

Abeillé, Anne, Lionel Clément, and François Toussenel. 2003. Building a treebank for French. In Anne Abeillé, editor, *Treebanks*. Kluwer.

Achterberg, Tobias. 2009. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41.

Attardi, Giuseppe and Massimiliano Ciaramita. 2007. Tree revision learning for dependency parsing. In *HLT-NAACL*, pages 388–395, Rochester, New York.

Bansal, Mohit and Dan Klein. 2011. Web-scale features for full-scale parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 693–702, Portland.

Bechet, Frederic and Alexis Nasr. 2009. Robust dependency parsing for spoken language understanding of spontaneous speech. In *Proceedings of Interspeech*, pages 1039–1042, Brighton, UK.

Bikel, Daniel M. 2004. Intricacies of Collins' parsing model. *Computational Linguistics*, 30(4):479–511.

Bohnet, Bernd. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 89–97, Beijing.

Boullier, Pierre, Alexis Nasr, and Benoît Sagot. 2009. Constructing parse forests that include exactly the n-best PCFG trees. In *11th Conference on Parsing Technologies*, pages 117–128, Paris.

Brent, Michael. 1991. Automatic acquisition of subcategorization frames from untagged text. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 209–214, Berkeley, CA.

Candito, Marie, Benoît Crabbé, Pascal Denis, and François Guérin. 2009. Analyse syntaxique du français: des constituants aux dépendances. In *Proceedings of Traitement Automatique des Langues Naturelles*, Senlis, France.

Charniak, Eugene and Mark Johnson. 2005. Coarse-to-fine *n*-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 173–180, Michigan.

Church, K. W. and P. Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29.

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23, Madrid, Spain.

Collins, Michael and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Eisner, Jason. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in Probabilistic and Other Parsing Technologies*, pages 29–61, Springer.

Eisner, Jason M. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th Conference on Computational Linguistics-Volume 1*, pages 340–345, Copenhagen, Denmark.

Gildea, Daniel. 2001. Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 167–202, Pittsburgh.

Hall, Keith and Václav Novák. 2005. Corrective modeling for non-projective dependency parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 42–52, Vancouver, Canada.

Henestroza, Enrique Anguiano and Marie Candito. 2011. Parse correction with specialized models for difficult attachment types. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1222–1233, Edinburgh, Scotland.

Huang, L. and D. Chiang. 2005. Better *k*-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64, Vancouver.

Huang, Liang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594, Columbus.

Huang, Liang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 144–151, Prague.

Huang, Zhongqiang, Mary Harper, and Slav Petrov. 2010. Self-training with products of latent variable grammars. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 12–22, Cambridge, Massachusetts.

Hwa, Rebecca. 2004. Sample selection for statistical parsing. *Computational Linguistics*, 30(3):253–276.

Joshi, Aravind Krishna. 1985. *Tree Adjoining Grammars: How Much Context-Sensitivity is Required to Provide Reasonable Structural Descriptions?* Technical report.

Karp, Richard M. 1972. *Reducibility among Combinatorial Problems*. Springer.

Koo, Terry and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden.

Kübler, Sandra, Ryan McDonald, and Joakim Nivre. 2009. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127.

Lecerf, Yves. 1961. Une représentation algébrique de la structure des phrases dans diverses langues naturelles. *Compte Rendu de l'Acadadémie des Sciences de Paris*, 252:232–234.

Manning, Christopher. 1993. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 235–242, Columbus.

Marcus, M. P., M. A. Marcinkiewicz, and B. Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

Martins, André F. T., Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia.

Martins, André F. T., Noah A. Smith, and Eric P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 342–350, Suntec.

McDonald, R., K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Michigan.

McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2005b. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Michigan.

McDonald, Ryan and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 121–132, Prague.

McDonald, Ryan T. and Fernando C. N. Pereira. 2006. Online learning of

approximate dependency parsing algorithms. In *EACL*, pages 81–88, Trento, Italy.

Messiant, C., A. Korhonen, T. Poibeau, et al. 2008. Lexschem: A large subcategorization lexicon for French verbs. In *Proceedings of the Language Resources and Evaluation Conference*, Marrakech, Morocco.

Mirroshandel, S.A. and A. Nasr. 2011. Active learning for dependency parsing using partially annotated sentences. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 140–149, Dublin.

Mirroshandel, Seyed Abolghasem, Alexis Nasr, and Joseph Le Roux. 2012. Semi-supervised dependency parsing using lexical affinities. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 777–785, Jeju Island, Korea.

Mirroshandel, Seyed Abolghasem, Alexis Nasr, and Benoit Sagot. 2013. Enforcing subcategorization constraints in a parser using sub-parses recombining. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 239–247, Atlanta.

Nakov, P. and M. Hearst. 2005. Using the Web as an implicit training set: application to structural ambiguity resolution. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing*, pages 835–842, Vancouver, Canada.

Napoles, Courtney, Matthew Gormley, and Benjamin van Durme. 2012a. Annotated English gigaword ldc2012t21. Linguistic Data Consortium, Philadelphia, PA.

Napoles, Courtney, Matthew Gormley, and Benjamin van Durme. 2012b. Annotated gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, pages 95–100, Montreal, Canada.

Nasr, A., F. Béchet, J. F. Rey, B. Favre, and Le Roux, J. 2011. MACAON: An NLP tool suite for processing word lattices. In *Proceedings of the ACL-HLT 2011 System Demonstrations*, pages 86–91, Portland.

Parker, Robert, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. English gigaword fifth edition, ldc2011t07. Linguistic Data Consortium, Philadelphia, PA.

Pitler, E., S. Bergsma, D. Lin, and K. Church. 2010. Using Web-scale N-grams to improve base NP parsing performance. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 886–894, Beijing, China.

Riedel, Sebastian and James Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 129–137, Sydney, Australia.

Rush, Alexander M. and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 498–507, Jeju Island, Korea.

Sánchez-Sáez, R., J. A. Sánchez, and J. M. Benedí. 2009. Statistical confidence measures for probabilistic parsing. In *Proceedings of the International Conference RANLP-2009*, pages 388–392, Borovets, Bulgaria.

Volk, M. 2001. Exploiting the WWW as a corpus to resolve PP attachment ambiguities. In *Corpus Linguistics*, pages 601–606.

Wessel, Frank, Ralf Schluter, Klaus Macherey, and Hermann Ney. 2001. Confidence measures for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 9(3):288–298.

Zhou, G., J. Zhao, K. Liu, and L. Cai. 2011. Exploiting Web-derived selectional preference to improve statistical dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1556–1565, Portland.