Dependency Parsing Resources for French: Converting Acquired Lexical Functional Grammar F-Structure Annotations and Parsing F-Structures Directly

Natalie Schluter
Dublin City University
Dublin 9, Ireland
nschluter@computing.dcu.ie

Josef van Genabith
Dublin City University
Dublin 9, Ireland
josef@computing.dcu.ie

Abstract

Recent years have seen considerable success in the generation of automatically obtained wide-coverage deep grammars for natural language processing, given reliable and large CFG-like treebanks. For research within Lexical Functional Grammar framework, these deep grammars are typically based on an extended PCFG parsing scheme from which dependencies are extracted. However, increasing success in statistical dependency parsing suggests that such deep grammar approaches to statistical parsing could be streamlined. We explore this novel approach to deep grammar parsing within the framework of LFG in this paper, for French, showing that best results (an f-score of 69.46) for the established integrated architecture may be obtained for French.

1 Introduction

Recent years have seen considerable success in the generation of automatically obtained wide-coverage deep grammars for natural language processing, given reliable and large CFG-like treebanks (for example, (Cahill et al., 2002; Guo et al., 2007; Chrupała and van Genabith, 2006)). For research within Lexical Functional Grammar (LFG) framework, these deep grammars are typically based on an extended PCFG parsing scheme from which dependencies are extracted. However, increasing success in statistical dependency parsing suggests that such deep grammar approaches to statistical parsing could be streamlined. In this paper, we explore this novel approach to deep grammar parsing within the framework of LFG in this paper, for French, showing that best results (an f-score of 69.46) for the established integrated architecture may be obtained for French.

This paper presents a *mise-en-scène* between theoretical dependency syntax and dependency parser practical requirements, an *entrée en scène* for f-structures in the literature for dependency parsing, an approach to representing f-structures in LFG as pseudo-projective dependencies, a first attempt to reconcile parsing LFG and dependency parsing, and, finally, the first treebank-based statistical dependency parsing results for French.

We begin with an brief introduction to LFG, followed by a presentation of the Modified French Treebank (the data source for this research), and an overview of previous parsing architecture of LFG f-structures (Section 2). Following this, we discuss LFG f-structure dependencies, comparing previously mentioned theoretical frameworks for statistical dependency parsing in the literature and showing their pseudo-projectivity (Section 3). In Section 4 we describe the data conversion involved in this research, and we end with the presentation of results of our experiments and a brief discussion (Section 5).

2 Preliminaries

2.1 Lexical Functional Grammar Basics

LFG is a constraint-based theory of language, whose basic architecture distinguishes two levels of syntactic representation : *c-structure* (constituent structure) and *f-structure* (functional structure)—c-structures corresponding to traditional constituent tree representation, and f-structures to a traditional dependency representation in the form of an attribute value matrix.[1]

Like any attribute-value matrix, f-structures are the minimal solution to a set of functional equations such as $(f\ a) = v$, where f is an f-structure, a is some attribute, and v is the value taken by that attribute, possibly another f-structure.

[1] A detailed introduction to LFG may be found in (Dalrymple, 2001).

Kristiina Jokinen and Eckhard Bick (Eds.)
NODALIDA 2009 Conference Proceedings, pp. 166–173

These two levels of representation (f-structure and c-structure), for a given phrase, are explicitly related by a structural mapping, called the *f-description*, often denoted by ϕ, which maps c-structure nodes to f-structure nodes.

In the LFG framework, this mapping may be given by functional annotations inserted into the c-structure tree, as in Figure 1.

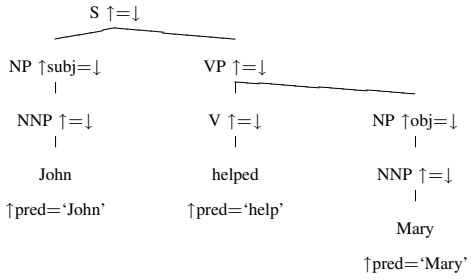

Figure 1: Annotated C-structure for *John helped Mary*.

The metavariables ↑ and ↓ refer to the f-structure of the mother node and that of the node itself, respectively. So that if node n, is annotated ↑=↓, then n's f-structure is mapped to the same f-structure as n's mother's f-structure. Also, if n has the annotation ↑obj=↓, this means that the f-structure associated with n is mapped to the value of the mother's f-structure obj attribute. LFG also has equations for members of sets, such as ↓∈↑adjunct, which states that the node's f-structure is mapped to an element of the mother's ADJ attribute.

2.2 Modified French Treebank

For this research, the treebank adopted is the Modified French Treebank (MFT) (Schluter and van Genabith, 2007). One important feature of the MFT is the extended function tag set, which includes function path tags. Consider the sentence in Example (1), taken directly from the MFT, whose tree structure is given in Figure 2.

(1) C'est [...] l'URSS [...] qui se
It is [...] the USSR [...] who herself
trouve prise
finds taken
'It is the USSR that finds itself trapped'[2]

In this example, the Srel constituent takes the functional path tag SUJ.MOD, representing the fact that Srel has the function MOD and is dependent on the constituent whose function is SUJ.

[2]Sentence 8151, file flmf3_08000_08499ep.xd.cat.xml.

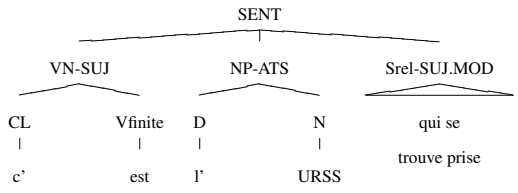

Figure 2: MFT representation of Example (1).

For this research, we followed the same random partition as (Schluter and van Genabith, 2008) for the training set (3800 sentences), test set (430 sentences), and development set (509 sentences).

2.3 Previous Parsing Architectures of LFG F-Structures

Previously, the technology for treebank-based acquisition of multilingual LFG probabilistic parsing resources consisted of three main stages, the basic input for which is a CFG-type treebank. These stages include the construction and application of an f-structure annotation algorithm combined with satisfiability verification, subcategorisation frame extraction, and long-distance dependency extraction. Given the resources produced in these initial stages, four CFG-based probabilistic parsing architectures were developed. Figure 3 shows these parsing architectures, with, in bold grey, the additional probabilistic dependency-based integrated architecture that is being presented here. (For more information on these former CFG-based probabilistic parsing architectures for French, see (Schluter and van Genabith, 2008)).

2.3.1 F-Structure Annotation Algorithm

For the creation of the dependency bank for French, to be used as training material, we will use f-structures. In the LFG framework, f-structures may be fully specified by f-structure annotated c-structures. The f-structure annotation algorithm outlined in (Schluter and van Genabith, 2008) will be adopted here, to obtain f-structure annotated c-structures. For French, it uses head-finding principles for a given constituent, to annotate MFT trees according to one of four modules:

1. **LFG Conversion Module**: MFT functional tags are directly translated into LFG function equations with respect to the constituent under consideration. For example, the

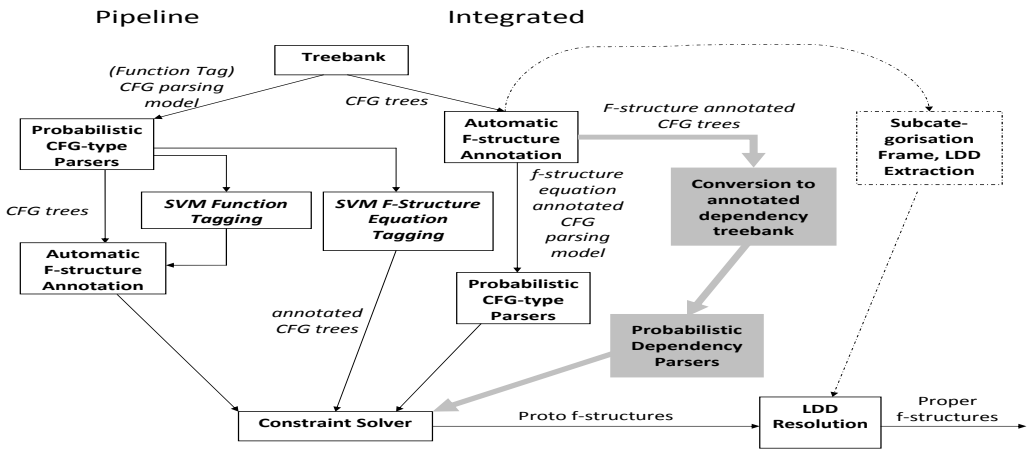

Figure 3: Overview of treebank-based LFG parsing architectures. The proposed dependency-based architecture being outlined in this paper is in bold grey.

functional tag ATO may be mapped to ↑-xcomp=↓, ↑-obj=↓-subj.

2. **Right-Left Annotation Module**: Constituent daughters are annotated with respect to the location of the constituent head. The constituent head is simply annotated as the predicate (↑-pred=*lemma*), or the governor of the predicate (↑=↓).

3. **Verb Combinatorics Module**: Combinations of different verb complexes in the MFT's constituent VN are resolved to predicates with corresponding compound tenses, for a monoclausal f-structural representation of verb phrases.

4. **Catch-all and Clean-up Module**: Any correction of detected overgeneralisations or miscellaneous annotations (such as sentence type) are carried out.

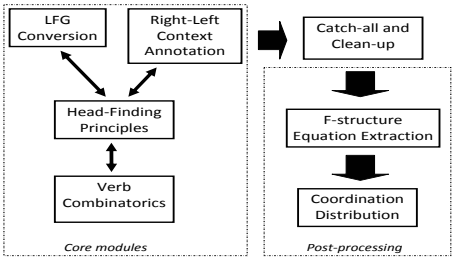

Figure 4: French Annotation Algorithm.

The f-structure annotation algorithm French was evaluated against a hand-corrected gold standard dependency bank. It achieves 98.4% coverage, with a best preds-only f-score of 99.63 (Table 1).

coord dist	precision	recall	f-score
no	98.57	96.38	97.46
yes	99.49	99.77	99.63

Table 1: Preds-only f-structure annotation algorithm performance.

2.3.2 Previous LFG Parsing Results for French

In this paper, because dependency parsers parse relations between actual word-forms and not any other features, we consider only preds-only results for LFG f-structures. That is, we evaluate only along those branches of the f-structures that end in a predicate.

(Schluter and van Genabith, 2008) report parsing results for both the integrated and pipeline architectures, with a best preds-only f-score for the integrated architecture of 67.88.[3]

3 LFG F-structure Dependencies

In this section, we overview the different target frameworks for the conversion of CFG-like data into dependency tree data (Section 3.1). We then

[3]The pipeline architecture outperforms the integrated architecture in (Schluter and van Genabith, 2008).

consider projectivity in light of these conversions, and explain why projectivity need not be a problem in LFG dependency parsing as a result of its the property of pseudo-projectivity (Section 3.2).

3.1 A Comparison of Theoretical Frameworks

In the statistical dependency parsing literature, there are generally two sources of modern linguistic theoretical justification behind parsing models: the theoretical framework of the Meaning-Text Theory (Mel'čuk, 1998), and the annotation guidelines of the Prague Treebank (Hajič et al., 1999). Moreover, software converting phrase-structure style treebanks into dependencies for statistical dependency parsing usually quote these two annotation styles in the treatment of hard cases. Therefore, when statistically parsing LFG f-structures, it is vital to consider what sorts of dependencies existing dependency parsers were intended to parse.

Meaning-Text Theory (MTT) represents the syntactic organisation of sentences strictly by dependencies. Under this framework, syntax is separated into surface and deep syntactic dependency-based tree representations. The deep-syntactic structure of a sentence has nodes that are semantically full lemmata (full lexemes); abstraction is made of any auxiliary or structural lemmata at this level. Also, lemmata are subscripted by the grammatical information (grammemes) expressed by their associated word-form(s), not imposed by government and agreement. Arcs are labeled by a selection of around ten language-independent relations. On the other hand, the surface-syntactic structure of a sentence contains all lemmata of the sentence and its arcs are labeled with the names of language-specific surface-syntactic relations, each of which represents a particular construction of the language (Mel'čuk, 2003; Mel'čuk, 1998). Furthermore, communicative functions such as `topic` or `focus` are not associated with a pure syntactic structure in the Meaning-Text Theory (Mel'čuk, 2001).

The Prague Treebank (PT) annotation guidelines (Hajič et al., 1999) also distinguishes between two levels of dependency-based syntactic representation: analytical and tectogrammatical. These guidelines are written in the spirit of Functional Generative Description.[4] These two levels of syntactic representation roughly correspond to those of the Meaning-Text Theory—the analytical level corresponding to the surface-syntax of the MTT and the tectogrammatical level corresponding to the deep-syntactic level of the MTT (Žabokrtský, 2005). In the PT, word-forms have attributes for their lemmata as well as for grammatical and lexical information expressed morphologically. The syntactic structure of the treebank is given for the analytic level of representation, though work is under way on complementing this with a tectogrammatical level of representation (Sgall et al., 2004). Also similarly to the MTT, communicative structure is not associated with pure syntax in Functional Generative Description, and therefore does not figure among annotations defined for the PT.

LFG does not have a uniform dependency syntax, distinguishing between c-structure and f-structure. These two systems contain different sorts of information, represented by means of phrase-structure trees, for the c-structure, and dependency dags, for f-structures. The f-structure is an abstract functional syntactic representation of a sentence, thought to contain deeper or more language-independent information than the c-structure (Dalrymple, 2001).

There are several important ways in which f-structures differ from the tree-dependencies outlined by the literature on dependency syntax within the MTT framework or the annotation guidelines of the Prague Treebank. For instance, included in the f-structure is communicative information, such as `topic` and `focus`, that LFG theorists consider to be grammaticised or syntacticised components of *information structure*). This introduces the notion of long-distance dependencies. Moreover, subject and object-raising are represented with re-entrancies at the syntactic level in LFG. This creates dags rather than just dependency trees, since some grammatical functions share the same f-structure value; these shared f-structures are called *re-entrancies*. Also, f-structure syntax corresponds, in fact, to a sort of mix of surface and deep dependency MTT syntax (respectively, a mix of analytic and tectogrammatical syntax). Like a surface dependency syntax, some lemmata, like copular verbs, that are not semantically full, appear in f-structures. On the

[4] See, for example, (Hajičová and Sgall, 2003) for a discussion of dependency syntax according to Functional Generative Description.

other hand, like deep dependency syntax, some lemmata that are not semantically full are excluded (for example, for the monoclausal treatment of compound tenses of verbs).

Other differences between dependency structures may be found in the notions of grouping and sets. In particular, coordination receives different treatments that must be considered. According to the PT annotation guidelines, coordination is treated as sets (conjuncts are sister nodes, elements of a set of conjuncts). Also, every node of a dependency tree must be associated with a word-form, which makes the coordinating conjunction or punctuation the governor of the set. On the other hand, in the MTT, coordination has a cascaded representation, with the first conjunct as governor. To distinguish between modifiers or arguments of the first conjunct and those of the coordinated structure, MTT theorists resort to *grouping*: the first conjunct essentially forms a distinguished group with its modifiers and arguments, much like the notion of constituent (Mel'čuk, 2003). In this sense, the first conjunct grouping is really the governor of the coordination.[5] Also according to the MTT, every node of a dependency tree must be associated with a word-form. But in LFG this is not necessary, in particular, in the representation of both coordination; coordinated elements, like in the PT, are treated as sets. In dag form, it can be seen that these coordinated structures have a *null* governor; that is, they do not have a governor that corresponds to any word-form as the node has no label. Because today's statistical dependency parsers cannot handle null elements, some pre-processing will be needed to convert our LFG representation of coordination (Section 4.1).

Finally, f-structures may be specified in terms of annotated c-structures with the local metavariables $\uparrow$ and $\downarrow$, and grammatical function regular paths. This restricts the structure of dependencies actually occurring in LFG f-structure syntax, as we will show in Section 3.2.

3.2 The Breadth of Functional Equations in LFG

LFG's f-structures often have re-entrancies (or shared sub-f-structures)—two functional equations resolve to take the same (f-structure) value—making them dags, rather than simple dependency trees. In LFG, the term *functional uncertainty* describes the uncertainty in the resolution given a simple grammatical function, in the definition of the grammar. The set of options for resolution may be finite and given by a disjunction, in which case resolution is down a chain of f-structure nodes of bounded length, or (theoretically) infinite in which case they are given by a regular expression (including the Kleene star operator) and resolution is down a chain of f-structure nodes of unbounded length. We note, however, that in statistical parsing of f-structures, the functional uncertainty in the resolution of a grammatical function will never be infinite, since the data is finite.

3.2.1 Projectivity

Consider a labeled dependency tree (directed tree) $T = (V, E, L)$, where V is its set of vertices (or nodes), $E = \{(a, l, b) \mid a, b \in V, l \in L\}$ its set of directed edges, and L the set of labels for edges. If $e = (a, l, b) \in E$, we say that a *immediately dominates* b; in this case, we say that a is the governor of b, or that b is a dependent on a. We say that v_1 *dominates* v_n if there is a chain of arcs $e_1, e_2, \ldots, e_{n-1}$, such that $e_1 = (v_1, l_1, v_2), e_2 = (v_2, l_2, v_3), \ldots, e_{n-1} = (v_{n-1}, l_{n-1}v_n)$. In this case, we also say that v_n is a descendent of v_1 or that v_1 is an ancestor of v_n.

An *ordered tree* is a tree having a total order, $(V, \leq)$, over its nodes, which for dependency trees is just the linear order of the symbols (or natural language words) in the generated string. An edge $e = (a, b)$ covers nodes $v_1, v_2, \ldots, v_n$ if $a \leq v_1, \ldots, v_n \leq b$, or $b \leq v_1, \ldots, v_n \leq a$.

An edge, $e = (v_1, l, v_2)$, of a tree is said to be projective if and only for every vertex v covered by e, v is dominated by v_1. A tree T is projective if and only if all its edges are projective (Robinson, 1970). (Gaifman, 1965) explains that a projective dependency tree can be associated with a dependency tree whose constituents are the projections of the nodes of the dependency tree, showing that projectivity in dependency trees corresponds to constituent continuity in phrase-structure trees.

These definitions are easily extended to dags. However in the case of dags, there are sometimes two governors for a single node that must be considered. For f-structure dags, we must additionally consider the mixed surface/deep dependency structure: some lemmata do not appear in f-structures as predicates. For those f-structure dags

[5] Grouping may be indicated on labels (Nilsson et al., 2006).

for which there is a one-to-one correspondence between predicates and original word-forms, these extended definitions may easily be applied.

However, LFG's treatment of long-distance dependency resolution and of subject/object raising is non-projective, illustrate non-projective dags. For French, for example, an interesting non-projective structure is found in *en* pronouns and NP extraction.

Projectivity in dependency trees or dags is obviously a result of the definition of the generating dependency grammar. This is true also of cases that are not LFG re-entrancies. For example, (Johansson and Nugues, 2007) propose a conversion of the Penn Treebank into dependency trees that introduces more projective edges than the conversion proposed by (Yamada and Matsumoto, 2003; Nivre, 2006). In addition to long-distance dependencies, for example, their representation of gapping always introduces non-projective branches (Johansson and Nugues, 2007).

LFG is capable of locally representing non-projective dependencies in phrase structures, which should, by definition, be impossible. This is because the only types of non-projective dependencies theoretically represented in LFG are actually pseudo-projectivities.

3.2.2 Non-Projectivity and Pseudo-Projectivity

Dependency trees also model non-projective structures that have no correspondence with any constituent trees—that is, they may be non-projective. This added "increase" in power for dependency grammars is shown to be useful for syntactic representations of certain languages (for example, the cross-serial dependencies of Dutch). However, as (Kahane et al., 1998) explain, pseudo-projective dependency trees may be parsed as projective trees with the aid of a simple transformation.

Consider two non-projective labeled dependency trees, $T_1 = (V, E_1, L_1)$ and $T_2 = (V, E_2, L_2)$. T_2 is called a *lift* if one of the following conditions hold, for some $e = (a, l, b), e' = (b, l', c) \in E_1$.[6]

1. $E_2 = (E_1 - \{e, e'\}) \cup \{(a, l : l', c)\}, L_2 \subseteq L_1 \cup \{l : l'\}$, or
2. T_3 is a lift of T_1 and T_2 is a lift of T_3.

[6]This definition is equivalent to the one given in (Kahane et al., 1998), where a lift was defined as in terms of governance for *unlabeled* dependency trees.

Corresponding to item 1 of the above conditions, the action of creating the tree T_2 from T_1 by removing the edges e, e' and adding the edge e'', will be referred to as *lifting*. A labeled ordered dependency tree T is said to be *pseudo-projective* if there is some lift T' of T that is projective.

(Kahane et al., 1998) explain that (for unlabeled dependency trees) one may make these definitions meaningful through the specification of lifting rules of the form $LD \uparrow SG\ w\ LG$, meaning that a node of category LD can be lifted to its syntactic governor of category LG through a path consisting of nodes of category $C_1, \ldots, C_n$, where C_i is among a specific set of categories (labels) (L_w) for all $i \in \{1, \ldots, n\}$. Equivalently, for a labeled ordered dependency tree, the path w may be specified by a path of labels. In this sense, building a projective tree by means of lifting results in arcs with path labels. Projecting the nodes would result in a sort of annotated c-structure. In this sense, and making abstraction of any contractions resulting from the annotations $\uparrow{=}\downarrow$, lifting is the opposite of the correspondence ϕ from c-structure to f-structure.[7] Re-entrancies may simply be considered as complex labels. Let us call the transformation opposite to lifting a *de-contraction* (used to undo the lifting transformation). Since generating an f-structure from an annotated c-structure involves simple contractions of the form $\uparrow{=}\downarrow$ and de-contractions, all f-structures are at most pseudo-projective. That means, we do not have to worry about non-projective structures in the parsing of LFG dependencies in f-structures.

4 Transforming Annotated C-Structures into Dependency Trees

To generate dependency trees, rather than using f-structures, we start with annotated c-structures. The motivation for this choice is straightforward: we need only carry out a certain number of contractions for the equations $\uparrow{=}\downarrow$ in order to get a projective dependency tree (rather than just a pseudo-projective dependency tree on which we must perform lifts). Moreover, the association of labels for handling re-entrancies is sitting in the annotated tree and does not need to be recalculated. There are some problems that remain in the result.

[7](Kahane et al., 1998) remark that the idea of building a projective tree by means of lifting can be compared to the functional uncertainty of LFG.

Firstly, not every terminal will get have a predicate annotation. For example, in causative constructions like for the phrase *faire danser* ('to make dance'), the word-form *faire* would only be annotated with the feature $\uparrow$ `factive` $= +$, not as a predicate. These will simply be turned into f-structures rather than features, by changing annotations such as these to $\uparrow$ `factive:pred` $=$ '$faire$'.

Another problem is that coordination structures have no governor. These structures must be transformed. We choose to follow the annotation guidelines for the PT for this transformation, due to its similarity with LFG analyses. Some coordination structures of the treebank need alternative treatment. In particular, non-constituent coordination and unlike constituent coordination require analyses that are not covered in the those guidelines. We resort to extended dependency tag sets to treat these cases and retain projectivity.

4.1 Coordination Transformations

In general, coordination will be transformed in the spirit of the PT annotation guidelines. If there is a coordinating conjunction, then the last of these will be taken as the governor of the coordination, as in the Figure 5. In the case where there is no coordinating conjunction but there is coordination punctuation (like a comma or semicolon), we will take the last of these as the governor. Otherwise we will take the first conjunct of the coordination as the governor and revert to grouping through extended labels.

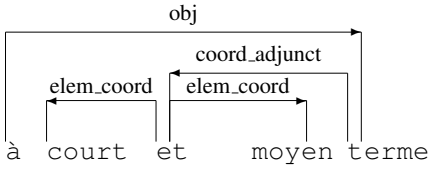

Figure 5: Dependency graph for *a court et moyen terme* ('short and mid-term').

For non-constituent coordination, the goal is twofold: (1) show that the different elements of each of the conjuncts belong together[8] and (2) show that they are missing something that is present in the first conjunct (done by the function tags). For this reason, the LFG analysis is ideal. However, a surface dependency analysis cannot do this; constituent structure is not simply dependency structure that projects lexical units to terminals. It shows groupings of elements based dependence on a item that is there or not. To do this, we use extended labels, forcing a "fake" lexical head.

5 Dependency Parsing Results

The parsing architecture works as follows. The annotation algorithm is applied to MFT trees, creating f-structure annotated trees that are then transformed into the projective depenendency representation described in Section 4, using the c-structure with the (only) f-structure equations. A dependency parser is then trained on this data, and the test set parsed. The parser output is then transformed back to f-structure equations, which are evaluated against the f-structure gold standard.

Two different dependency parsers were used for this research: MST parser (McDonald et al., 2005) and MALT parser (Nivre et al., 2006). Experiments were done with the simplified architecture (in which long-distance dependencies are given as complex path equations in training), and in the established architecture (with a separated long-distance dependency resolution task).[9] The results are given in Tables 2 and 3.

Parser	coord dist	precision	recall	f-score
MST	*no*	87.46	54.67	67.28
	yes	87.45	54.66	67.27
MALT	*no*	86.23	52.17	65.01
	yes	86.17	51.95	64.82

Table 2: Simplified Architecture Parsing Results

Parser	LDDs resolved	coord dist	precision	recall	f-score
MST	*no*	*no*	86.90	57.07	68.89
		yes	86.89	57.06	68.88
	yes	*yes*	86.48	58.03	69.46
MALT	*no*	*no*	85.98	51.13	64.13
		yes	86.02	50.9	63.96
	yes	*yes*	86.08	51.62	64.54

Table 3: Parsing Results with Long Distance Dependency Resolution

[8]The dependency treatment of coordination outlined by (Johansson and Nugues, 2007) for the treatment of gapping also introduced ambiguity for the case where there are more than two conjuncts; in this solution, they have removed the relation that the components of gapping are part of a same element/constituent.

[9]More information on the difference between these two architectures can be found in (Schluter and van Genabith, 2008).

We observe that best results are obtained by the MST parser when LDD recovery separated and coordination distribution is carried out.

6 Concluding Remarks

In this paper, we have shown that best statistical parsing results for French in the integrated LFG parsing architecture are achievable by extending this architecture for statistical dependency parsing. However, best results, in general are still obtained via the original PCFG based LFG parsing approach. Future work would look at extending the use of machine learning to approximate the integrated parsing architecture, which has been shown to improve results in the PCFG based LFG parsing approach.

Acknowledgments

This research was supported by Science Foundation Ireland GramLab grant 04/IN/I527.

References

A. Cahill, M. McCarthy, J. van Genabith, and A. Way. 2002. Automatic annotation of the penn treebank with lfg f-structure information. In A. Lenci, S. Montemagni, and V. Pirelli, editors, *Proceedings of the LREC 2002 workshop on Linguistic Knowledge Acquisition and Representation*, Paris. ELRA.

G. Chrupała and J. van Genabith. 2006. Improving treebank-based automatic lfg induction for spanish. In *Proc. of LFG06*, Konstanz, Germany.

Mary Dalrymple. 2001. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics*. Academic Press, San Diego.

Haim Gaifman. 1965. Dependency systems and phrase-structured systems. *Information and Control*, 8:304–337.

Y. Guo, J. van Genabith, and H. Wang. 2007. Treebank-based acquisition of lfg resources for chinese. In *Proc. of LFG07*, Stanford, CA.

J. Hajič, J. Panevová, E. Buráňová, Z. Urešová, and A. Bémová. 1999. Annotations at analytical level. instructions for annotators. Technical report, Prague.

E. Hajičová and P. Sgall, 2003. *Dependency Syntax in Functional Generative Description*, pages 570–592. Walter de Gruyter, Berlin and New York.

R. Johansson and P. Nugues. 2007. Extended constituent-to-dependency conversion. In *NODALIDA 2007 Conference Proceedings*, pages 105–112, Tartu, Estonia.

S. Kahane, A. Nasr, and O. Rambow. 1998. Pseudo-projectivity: a polynomially parsable non-projective dependency grammar. In *Proceedings of the 17th international conference on Computational linguistics*, pages 646–652.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proc. of ACL 2005*.

I. Mel'čuk. 1998. *Vers une linguistique Sens-Texte. Leçon inaugurale*. Collège de France, Paris.

I. Mel'čuk. 2001. *Communicative Organisation in Natural Language. The Semantic-Communicative Structure of Sentences*. Benjamins, Amsterdam and Philadelphia.

I. Mel'čuk, 2003. *Levels of Dependency in Linguistic Description: Concepts and Problems*, volume 1, pages 188–229. Walter de Gruyter, Berlin and New York.

J. Nilsson, J. Nivre, and J. Hall. 2006. Graph transformations in data-driven dependency parsing. In *ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 257–264.

J. Nivre, J. Hall, and J. Nilsson. 2006. Malt-parser: A data-driven parser generator for dependency parsing. In *Proceedings of LREC'06*.

J. Nivre. 2006. *Inductive Dependency Parsing*. Springer Verlag.

J. J. Robinson. 1970. Dependency structure and transformational rules. 46:259–285.

N. Schluter and J. van Genabith. 2007. Preparing, restructuring, and augmenting a french treebank: Lexicalised parsers or coherent treebanks? In *Proc. of the PACLING 2007*, Melbourne, Australia.

N. Schluter and J. van Genabith. 2008. Treebank-based acquisition of lfg parsing resources for french. In *Proc. of LREC 2008*, Marrakech, Morocco.

P. Sgall, J. Panevová, and E. Hajičová. 2004. Deep syntactic annotation: Tectogrammatical representation and beyond. In A. Meyers, editor, *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 32–38, Boston, MASS. Association for Computational Linguistics.

Z. Žabokrtský. 2005. Resemblances between meaning-text theory and functional generative description. In J.D. Apresjian and L.L. Iomdin, editors, *Proceedings of the 2nd International Conference of Meaning-Text Theory*, pages 549–557, Moscow, Russia. Slavic Culture Languages Publishers House.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206, Nancy, France.

ISSN 1736-6305 Vol. 4
http://hdl.handle.net/10062/9206