Antecedent Recovery: Experiments with a Trace Tagger

Péter Dienes and Amit Dubey
Department of Computational Linguistics
Saarland University
PO Box 15 11 50
66041 Saarbrücken, Germany
{dienes,adubey}@coli.uni-sb.de

Abstract

This paper explores the problem of finding non-local dependencies. First, we isolate a set of features useful for this task. Second, we develop both a two-step approach which combines a trace tagger with a state-of-the-art lexicalized parser and a one-step approach which finds non-local dependencies while parsing. We find that the former outperforms the latter because it makes better use of the features we isolate.

1 Introduction

Many broad-coverage statistical parsers (Charniak, 2000; Collins, 1999; Bod, 2001) are not able to give a full interpretation for sentences such as:

(1) *It is difficult to guess what she wants to buy.*

Building the semantic interpretation of this sentence requires recovering three non-local relations: (i) the object of *buy* is *what*;[1] (ii) the subject of *buy* is *she*; and (iii) *guess* does not have a subject in the sentence.

Three approaches have been proposed to detect such relations: (i) post-processing the output of a parser not designed to detect extraction sites (Johnson, 2002); (ii) integrating antecedent recovery into the parser (henceforth *in-processing*) by either enriching a syntactically simple model (Collins, 1999) or using a more powerful syntactic framework (Clark et al., 2002; Riezler et al., 2002); and (iii) detecting non-local dependencies as a pre-processing step before parsing (Dienes and Dubey, 2003).

While the pre-processing approach is reported to give state-of-the-art performance using unlexicalized parsers, it has not been tested using lexicalized models. Our main claim is that that the pre-processing approach, coupled with a *lexicalized* parser outperforms both state-of-the-art post-processing and in-processing. However, we show that Model 3 of Collins (1999) can be generalized to handle all types of long-distance dependencies with performance close to the pre-processing architecture.

A general contribution of this paper is that it gives important insights about the nature of the problem. Recovering non-local semantic relations is regarded to be a difficult problem. The successes (and failures) of the simple architecture outlined here help determine what features are to be incorporated into a parser in order to improve recovery of non-local dependencies.

The overall organization of the paper is as follows. First, Section 2 sketches the material we use for the experiments in the paper. In Section 3, we discuss a finite-state system, a *trace tagger*, that detects extraction sites without knowledge of phrase-structure and we isolate important cues for the task. Section 4 combines the trace tagger with a parser in order to recover antecedents. Finally, in Section 5, we investigate whether and how detection of extraction sites and antecedent recovery can be integrated into a lexicalized stochastic parser.

[1] Collins (1999) can handle this case (Model 3).

Type	Freq.	Explanation	Example
NP–NP	987	controlled NP-traces	Sam was seen *
WH–NP	438	NP-traces of A′-movement	the woman who you saw *T*
PRO–NP	426	uncontrolled PROs	* to sleep is nice
COMP–SBAR	338	empty complementizer (*that*)	Sam said 0 Sasha snores
UNIT	332	empty units	$ 25 *U*
WH–S	228	trace of topicalized sentence	Sam had to go, Sasha said *T*
WH–ADVP	120	traces of WH adverbs	Sam told us how he did it *T*
CLAUSE	118	trace of a moved SBAR	Sam had to go, Sasha said 0
COMP–WHNP	98	empty WH-complementizer	the woman 0 we saw *T*
ALL	3310		

Table 1: Most frequent types of EEs in Section 0.

2 Data

In the experiments we use the same training, test, and development data as in Dienes and Dubey (2003), where non-local dependencies are annotated with the help of empty elements (EEs) co-indexed with their controlling constituents (if any). The most frequent types of EEs are summarized in Table 1. Thus, the example sentence (1) will get the annotation:

(2) *It is difficult* PRO-NP *to guess what she wants* NP-NP *to buy* WH-NP.

For the parsing and antecedent recovery experiments, in the case of WH-traces (WH–$\cdots$) and controlled NP-traces (NP–NP), we follow the standard technique of marking nodes dominating the empty element up to but not including the parent of the antecedent as defective (missing an argument) with a gap feature (Gazdar et al., 1985; Collins, 1999). Furthermore, to make antecedent co-indexation possible with many types of EEs, we generalize Collins' approach by enriching the annotation of non-terminals with the *type* of the EE in question (eg. WH–NP), using different gap+ features (gap+WH-NP; c.f. Figure 1). The original non-terminals augmented with gap+ features serve as new non-terminal labels. Note, however, that not all EEs have antecedents. In these cases, the gap+ feature does not show up in the dominating non-terminal (Figure 2).

3 Detecting empty elements

Previous work (Dienes and Dubey, 2003) shows that detecting empty elements can be performed fairly reliably before parsing using a trace tagger, which tags words with information on EEs immediately preceding them. For example, the first occurrence of the word *to* in our example sentence (2) gets the tag EE=TT-NP, whereas the word *wants* is tagged as having no EE. The trace tagger uses three main types of features: (i) combination of POS tags in a window of five words around the EEs; (ii) lexical features of the words in a window of three lexical items; and (iii) long-distance cues (Table 2). An EE is correctly detected if and only if (i) the label matches that of the gold standard and (ii) it occurs between the same words. Dienes and Dubey (2003) report 79.1% labeled F-score on this evaluation metric, the

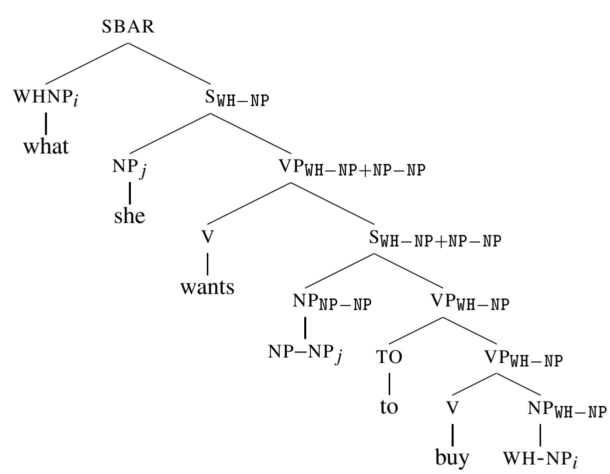

Figure 1: Threading gap+WH-NP and gap+NP-NP.

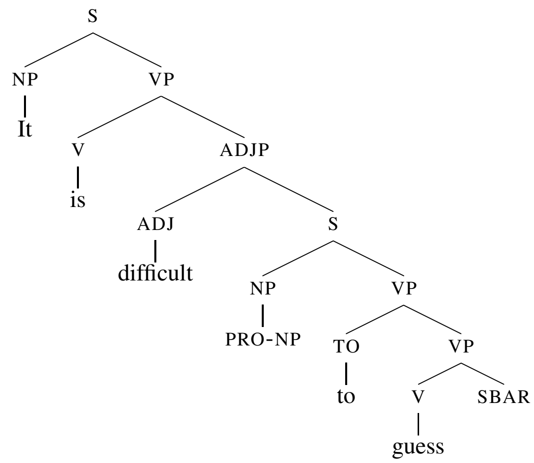

Figure 2: Representing EEs without antecedents.

best published result on the EE detection task.

While Dienes and Dubey (2003) report overall scores, they do not evaluate the relative importance of the features used by the tagger. This can be achieved by testing how the model fares if only a subset of the features are switched on (performance analysis). Another way to investigate the problem is to analyze the average weight and the activation frequency of each feature type.

According to the performance analysis, the most important features are the ones encoding POS-information. Indeed, by turning only these features on, the accuracy of the system is already fairly high: the labeled F-score is 71.2%. A closer look at the feature weights shows that the right context is slightly more informative than the left one. Lexicalization of the model contributes further 6% to the overall score (the following word being slightly more important than the preceding one), whereas the features capturing long-distance cues only improve the overall score by around 2%. Interestingly, long-distance features get higher weights in general, but their contribution to the overall performance is small since they are rarely activated. Finally, the model with only lexical features performs surprisingly well: the labeled F-score is 68.9%, showing that a very small window already contains valuable information for the task.

In summary, the most important result here is that a relatively small window of up to five words contains important cues for detecting EEs.

4 Antecedent recovery

Antecedent recovery requires knowledge of phrase structure, and hence calls for a parsing component. In this section, we show how to recover the antecedents given a parse tree, and how to incorporate information about EE-sites into the parser.

4.1 Antecedent recovery algorithm

The main motivation for the introduction of `gap+` variables is that they indicate a path from the EE to the antecedent. In case of a non-binary-branching grammar, however, this path only determines the node immediately dominating the antecedent, but does not indicate the child the EE should be co-indexed with. Moreover, a node might contain several `gap+` variables, which further complicates antecedent recovery, even in the case of perfect trees. This calls for a sophisticated algorithm to recover antecedents.

```
1  foreach gap
2    do find_antecedent("gap", gap);
3
4  proc find_antecedent("gap", node) ≡
5    var par = node.parent;
6      p = # of gap+ features of type "gap" on par;
7      ch =sum of the gap+ features of type "gap" on
8            par.children;
9      node.remove_one_gap("gap");
10     if p < ch
11       then Drop the gap here
12              ante =leftmost non-adjunct of par.children
13                     allowed by "gap"(≠ node);
14              return ante if ante;
15              ante =leftmost child of par.children
16                     allowed by "gap"(≠ node);
17              return ante if ante;
18              return nil;
19        else Pass up the tree recursively
20              find_antecedent("gap", par).
```

Figure 3: The antecedent recovery algorithm.

The algorithm, presented in Figure 3, runs after the best parse has been selected. It works in a bottom-up fashion, and for each empty node the main recursive function *find_antecedent* is called separately (lines 1 and 2). At every call, the number of `gap+` variables of type "`gap`" are calculated for the parent *par* of the current node *node* (p; line 6) and for all the children (ch; line 7). If the parent has at least as many unresolved `gap+` variables as its children, we conclude that the current EE is re-

Target	Matching regexp	Explanation
NP–NP	BE RB* VBN __	passive
{NP–NP, PRO-NP}	__ RB* *to* RB* VB	to-infinitive
	N [,:] __ RB* VBG	gerund
COMP–SBAR	(V | ,) __ *!that** (MD | V)	lookahead for *that*
WH–NP	!IN {WP, WDT, COMP–WHNP} !WH–NP* V __	lookback for pending WHNPs
WH–ADVP	WRB !WH–ADVP* V !WH–ADVP* __ [.,:]	lookback for pending WHADVP before a verb
UNIT	$ CD* __	$ sign before numbers

Table 2: Non-local binary feature templates; the EE-site is indicated by __

solved further up in the tree and call the same algorithm for the parent (line 20). If, however, the parent has fewer unresolved gaps ($p < ch$), the antecedent of the EE is among the children. Thus the algorithm attempts to find this antecedent (lines 11–18). For an antecedent to be selected, the syntactic category must match, i.e. an NP–NP must resolve to a NP. The algorithm searches from left to right for a possible candidate, preferring non-adjuncts over adjuncts. The node found (if any) is returned as the antecedent for the EE. Finally, note that in line 9, we have to remove the threaded gap+ feature in order to avoid confusion if the same parent is visited again while resolving another EE.

Although the algorithm is simple and works in a greedy manner, it does perform well. Tested on the gold standard trees containing the empty nodes without antecedent co-reference information, it is able to recover the antecedents with an F-score of 95% (c.f. Section 4.3).

4.2 Method

Antecedent recovery is tested using two parsers: an unlexicalized PCFG (Dienes and Dubey, 2003) and a lexicalized parser with near state-of-the-art performance (Collins, 1999). Both parsers treat EEs as words. In order to recover antecedents, both were modified to thread gap+ variables in the nonterminals as described in Section 2.

Each parser is evaluated in two cases: (i) an upper bound case which uses the perfect EEs of the treebank (henceforth PERFECT) and (ii) a case that uses EEs suggested by the finite-state mechanism (henceforth TAGGER). In the TAGGER case, the parser simply takes the hypotheses of the finite-state mechanism as true.

Condition		Bracketing	Antecedent recovery
PERFECT	UNLEX	78.5%	91.4%
	LEX	88.6%	93.3%
TAGGER	UNLEX	76.3%	72.6%
	LEX	86.4%	**74.6%**
Johnson		89.1%	68.0%

Table 3: F-Scores for parsing and antecedent recovery on Section 23.

4.3 Evaluation

We evaluate on all sentences in the test section of the treebank. As with trace detection, we use the measure introduced by Johnson (2002). This metric works by treating EEs and their antecedents as four-tuples, consisting of the type of the EE, its location, the type of its antecedent and the location(s) (beginning and end) of the antecedent. An antecedent is correctly *recovered* if all four values match the gold standard. We calculate the precision, recall, and F-score; however for brevity's sake we only report the F-score for most experiments in this section.

In addition to antecedent recovery, we also report parsing accuracy, using the bracketing F-Score, the combined measure of PARSEVAL-style labeled bracketing precision and recall (Magerman, 1995).

4.4 Results

The results of the experiments are summarized in Table 3. UNLEX and LEX refer to the unlexicalized and lexicalized models, respectively. In the upper-bound case, PERFECT, the F-score for antecedent recovery is quite high in both the unlexicalized and lexicalized cases: 91.4% and 93.3%.

Type	Prec. Here	Rec. Here	*F*-score Here	*F*-score Johnson
OVERALL	81.5%	68.7%	**74.6%**	68.0%
NP–NP	74.3%	67.4%	**70.7%**	60.0%
WH–NP	91.0%	74.5%	**82.0%**	80.0%
PRO–NP	68.7%	70.4%	**69.5%**	50.0%
COMP–SBAR	93.8%	78.6%	85.5%	**88.0%**
UNIT	99.1%	92.5%	**95.7%**	92.0%
WH–S	86.3%	82.8%	84.5%	**87.0%**
WH–ADVP	74.5%	42.0%	53.6%	**56.0%**
CLAUSE	80.4%	68.3%	**73.8%**	70.0%
COMP–WHNP	67.2%	38.3%	**48.8%**	47.0%

Table 4: Comparison of our antecedent recovery results with the lexicalized parser and Johnson's (2002).

Johnson (2002)'s metric includes EE without antecedents. To test how well the antecedent-detection algorithm works, it is useful, however, to count the results of only those EEs which have antecedents in the tree (NP–NP, PSEUDO attachments, and all WH traces). In these cases, the unlexicalized parser has an F-score of 70.4%, and the lexicalized parser 83.9%, both in the PERFECT case.

In the TAGGER case, which is our main concern, the unlexicalized parser achieves an F-score of 72.6%, better than the 68.0% reported by Johnson (2002). The lexicalized parser outperforms both, yielding results of F-score of 74.6%.

Table 4 gives a closer look at the antecedent recovery score for some common EE types using the lexicalized parser, also showing the results of Johnson (2002) for comparison.

4.5 Discussion

The pre-processing system does quite well, managing an F-score 6.6% higher than the post-processing system of Johnson (2002). However, while the lexicalized parser performs better than the unlexicalized one, the difference is quite small: only 2%. This suggests that many of the remaining errors are actually in the pre-processor rather than in the parser. Two particular cases of interest are NP–NPs and PRO–NPs. In both cases, a NP is missing, often in a to-infinitival clause. The two are only distinguished by their antecedent: NP–NP has an antecedent in the tree, while PRO–NP has none. The lexicalized parser has, for most types of EEs, quite high antecedent detection results, but the difficulty in telling the difference between these two cases results in low F-scores for antecedent recovery of NP–NP and PRO–NP, despite the fact that they are among the most common EE types. Even though this is a problem, our system still does quite well: 70.4% for NP–NP, and 69.5% for PRO–NP compared to the 60.0% and 50.0% reported by Johnson (2002).

Since it appears the pre-processor is the cause of most of the errors, in-processing with a state-of-the-art lexicalized parser might outperform the pre-processing approach. In the next section, we explore this possibility.

5 Detecting empty elements in the parser

Having compared pre-processing to post-processing in the previous section, in this section, we consider the relative advantages of pre-processing as compared to detecting EEs while parsing, with both an unlexicalized and a lexicalized model.

In making the comparison between detecting EEs during pre-processing versus parsing, we are not only concerned with the accuracy of parsing, EE detection and antecedent recovery, but also with the running time of the parsers. In particular, Dienes and Dubey (2003) found that detecting EEs is infeasible with an unlexicalized parser: the parser was slow and inaccurate at EE detection.

Recall that the runtime of many parsing algorithms depends on the size of the grammar or the number of nonterminals. The unlexicalized CYK parser we use has a worst-case asymptotic runtime of $O(n^3N^3)$ where n is the number of words and N is the number of nonterminals. Collins (1999) reports a worst-case asymptotic runtime of $O(n^5N^3)$ for a lexicalized parser.

The $O(N^3)$ bound becomes important when the parser is to insert traces because there are more nonterminals. Three factors contribute to this larger nonterminal set: (i) nonterminals are augmented with EE types that contain the parent node of the EE (i.e. S may become S_{WH-NP}, S_{WH-PP}, etc.) (ii) we must include combinations of EEs as nonterminals may dominate more than one unbound EE (i.e. $S_{NP-NP+WH-NP}$) and (iii) a single nonterminal may be repeated in the presence of co-ordination (i.e. $\mathrm{S}_{\mathtt{NP-NP+NP-NP}}$). These three factors greatly increase the number of nonterminals, potentially reducing the efficiency of a parser that detects EEs. On the other hand, when EE-sites are pre-determined, the effect of the number of nonterminals on parsing speed is moot: the parser can ignore large parts of the grammar.

In this section, we empirically explore the relative advantages of pre-processing over in-processing, with respect to runtime efficiency and the accuracy of parsing and antecedent recovery.

5.1 Method

As in Section 4, we use the unlexicalized parser from Dienes and Dubey (2003), and as a lexicalized parser, an extension of Model 3 of Collins (1999). While Model 3 inserts WH–NP traces, it makes some assumptions that preclude it from being used here directly:

(i) it cannot handle multiple *types* of EEs;

(ii) it does not allow multiple *instances* of EEs at a node;

(iii) it expects all EEs to be complements, though some are not (e.g. WH–ADVP);

(iv) it expects all EEs to have antecedents, though some do not (e.g. PRO–NP);

(v) it cannot model EEs with dependents, for example COMP–. . . .

Hence, Model 3 must be generalized to other types of discontinuities. In order to handle the first four problems, we propose generating 'gap-categorization' frames in the same way as subcategorization frames are used in the original model. We do not offer a solution to the final problem, as the syntactic structure (usually the unary production SBAR → S) will identify these cases.

After calculating the probability of the head (with its gaps), the left and right gapcat frame are generated independently of each other (and of the subcat frames). For example, the probability for the rule:

$$\begin{array}{l}\mathrm{VP}\ (to)\ (\texttt{+gap=}\{\texttt{WH-NP}\}) \rightarrow \\ \quad \mathrm{TO}\ (to)\ (\texttt{+gap=}\{\})\quad \mathrm{VP}\ (buy)\ (\texttt{+gap=}\{\texttt{WH-NP}\})\end{array}$$

Condition	Relative # of Nonterminals	Relative Parsing Time	Missed Parses
NOTRACE	1.00	1.00	0.2%
WH–NP	1.63	2.17	10.3%
PRO&WH	7.15	3.58	35.1%
TAGGER	7.15	1.49	1.3%

Table 5: INSERT model unlexicalized parsing results on Section 23.

is generated as:

$$\begin{array}{l}P_h(\mathrm{TO}\,|\,\mathrm{VP},\mathrm{TO},to)\times\\ P_{RGC}(\{\texttt{WH-NP}\}\,|\,\mathrm{VP},\mathrm{TO},to)\times P_{LGC}(\{\}\,|\,\mathrm{VP},\mathrm{TO},to)\times\\ P_{RC}(\{\mathrm{VP\text{-}C}\}\,|\,\mathrm{VP},\mathrm{TO},to)\times P_{LC}(\{\}\,|\,\mathrm{VP},\mathrm{TO},to)\times\\ P_r(\mathrm{VP\text{-}C}_{\texttt{WH-NP}}(buy)\,|\,\mathrm{VP},\mathrm{TO},to,\{\mathrm{VP\text{-}C}\},\{\texttt{WH-NP}\})\times\\ P_r(\mathrm{STOP}\,|\,\mathrm{VP},\mathrm{TO},to,\{\},\{\})\times\\ P_l(\mathrm{STOP}\,|\,\mathrm{VP},\mathrm{TO},to,\{\},\{\})\end{array}$$

Generating the actual EE is done in a similar fashion: the EE cancels the corresponding 'gapcat' requirement. If it is a complement (e.g. WH–NP), it also removes the corresponding element from the subcat frame. The original parsing algorithm was modified to accommodate 'gapcat' requirements and generate multiple types of EEs.

We compare the parsing performance of the two parsers in four cases: the NOTRACE model which removes all traces from the test and training data, the TAGGER model of Section 4, and two cases where the parser inserts EEs (we will collectively refer to these cases as the INSERT models). In order to show the effects of increasing the size of nonterminal vocabulary, the first INSERT model only considers one EE type, WH–NP while the second (henceforth PRO&WH) considers all WH traces as well as NP–NP and PRO–NP discontinuities.

5.2 Results

The results of the unlexicalized and lexicalized experiments are summarized in Tables 5 and Table 6, respectively. The tables compare relative parsing time (slowdown with respect to the NOTRACE model), and in the lexicalized case, PARSEVAL-style bracketing scores. However, in the case of the unlexicalized model, the increasing number of

Condition	Relative # of Nonterminals	Relative Parsing Time	Bracketing
NOTRACE	1.00	1.00	88.0%
WH–NP	1.63	1.07	87.4%
PRO&WH	7.15	1.33	86.6%
TAGGER	7.15	0.95	86.4%

Table 6: INSERT model lexicalized parsing results on Section 23.

Type	EE detection		Antecedent rec.	
	parser	*tagger*	*parser*	*tagger*
NP–NP	80.4%	83.5%	70.3%	70.7%
WH–NP	81.5%	83.2%	80.2%	82.0%
PRO–NP	64.5%	69.5%	64.5%	69.5%
WH–S	92.0%	92.8%	82.2%	84.5%
WH–ADVP	57.9%	59.5%	53.0%	53.6%

Table 7: Comparison of pre-processing with lexicalized in-processing (F-scores).

missed parses precludes straightforward comparison of bracketing scores, therefore we report the percentage of sentences where the parser fails. In the case of the lexicalized parser, less than 1% of the parses are missed, hence the comparisons are reliable. Finally, we compare EE detection and antecedent recovery F-scores of the TAGGER and the PRO&WH models for the overlapping EE types (Table 7).

5.3 Discussion

As noted by Dienes and Dubey (2003), unlexicalized parsing with EEs does not seem to be viable without pre-processing. However, the lexicalized parser is competitive with the pre-processing approach.

As for the bracketing scores, there are two interesting results. First, lexicalized models which handle EEs have lower bracketing scores than the NOTRACE model. Indeed, as the number of EEs increases, so does the number of nonterminals, which results in increasingly severe sparse data problem. Consequently, there is a trade-off between finding local phrase structure and long-distance dependencies.

Second, comparing the TAGGER and the PRO&WH models, we find that the bracketing results are nearly identical. Nonetheless, the PRO&WH model inserting EEs can match neither the accuracy for antecedent recovery nor the time efficiency of the pre-processing approach. Thus, the results show that treating EE-detection as a pre-processing step is beneficial to both to antecedent recovery accuracy and to parsing efficiency.

Nevertheless, pre-processing is not necessarily the only useful strategy for trace detection. Indeed, by taking advantage of the insights that make the finite-state and lexicalized parsing models successful, it may be possible to generalize the results to other strategies as well. There are two key observations of importance here.

The first observation is that lexicalization is very important for detecting traces, not just for the lexicalized parser, but, as discussed in Section 3, for the trace-tagger as well. The two models may contain overlapping information: in many cases, the lexical cue corresponds to the immediate head-word the EE depends on. However, other surrounding words (which frequently correspond to the head-word of grandparent of the empty node) often carry important information, especially for distinguishing NP–NP and PRO–NP nodes.

Second, local information (i.e. a window of five words) proves to be informative for the task. This explains why the finite-state tagger is more accurate than the parser: this window *always* crosses a phrase boundary, and the parser cannot consider the whole window.

These two observations give a set of features that seem to be useful for EE detection. We conjecture that a parser that takes advantage of these features might be more accurate in detecting EEs while parsing than the parsers presented here. Apart from the pre-processing approach presented here, there are a number of ways these features could be used:

1. in a pre-processing system that only detects EEs, as we have done here;

2. as part of a larger syntactic pre-processing system, such as supertagging (Joshi and Bangalore, 1994);

3. with a more informative beam search (Charniak et al., 1998);

4. or directly integrated into the parsing mechanism, for example by combining the finite-state and the parsing probability models.

6 Conclusions

One of the main contributions of this paper is that a two-step pre-processing approach to finding EEs outperforms both post-processing and in-processing. We found the pre-processing technique was successful because it used features not explicitly incorporated into the other models.

Furthermore, we found that the result presented in Dienes and Dubey (2003), i.e. pre-processing is better for antecedent recovery than unlexicalized in-processing, also holds when comparing lexicalized models. However, comparing the lexicalized pre-processing system to the unlexicalized one, we find that although lexicalization results in much better trees, there is only a slight improvement in antecedent recovery.

Third, we present a generalization of Model 3 of Collins (1999) to handle a broader range of EEs. While this particular model was not able to outperform the pre-processing method, it can be further developed into a parsing model which can handle non-local dependencies by incorporating the local cues we found relevant.

In particular, a local window of five words, accompanied by the `gap+` threads proved to be crucial. Thus we claim that, in order to detect long-distance dependencies, a robust stochastic parser should integrate lexical information as well as local cues cutting across phrase boundaries by either incorporating them into the probability model or using them in the beam-search.

Acknowledgements

The authors would like to thank Jason Baldridge, Matthew Crocker, Geert-Jan Kruijff, Shravan Vasishth and the anonymous reviewers for their invaluable suggestions and comments.

References

Rens Bod. 2001. What is the minimal set of fragments that achieves maximal parse accuracy? In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics and the 10th Conference of the European Chapter of the Association for Computational Linguistics*, Toulouse, France.

Eugene Charniak, Sharon Goldwater, and Mark Johnson. 1998. Edge-based best-first chart parsing. In *Proceedings of the 14th National Conference on Artificial Intelligence*, Madison, WI.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Conference of North American Chapter of the Association for Computational Linguistics*, Seattle, WA.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Péter Dienes and Amit Dubey. 2003. Deep syntactic processing by combining shallow methods. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan.

Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. 1985. *Generalized Phase Structure Grammar*. Basil Blackwell, Oxford, England.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA.

Aravind K. Joshi and Srinivas Bangalore. 1994. Complexity of descriptives–supertag disambiguation or almost parsing. In *Proceedings of the 1994 International Conference on Computational Linguistics*, Kyoto, Japan.

David Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, Cambridge, MA.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA.