Fully Parsing the Penn Treebank

Ryan Gabbard **Mitchell Marcus**
Department of
Computer and Information Science
University of Pennsylvania
{gabbard,mitch}@cis.upenn.edu

Seth Kulick
Institute for Research in
Cognitive Science
University of Pennsylvania
skulick@cis.upenn.edu

Abstract

We present a two stage parser that recovers Penn Treebank style syntactic analyses of new sentences including skeletal syntactic structure, and, for the first time, both function tags and empty categories. The accuracy of the first-stage parser on the standard Parseval metric matches that of the (Collins, 2003) parser on which it is based, despite the data fragmentation caused by the greatly enriched space of possible node labels. This first stage simultaneously achieves near state-of-the-art performance on recovering function tags with minimal modifications to the underlying parser, modifying less than ten lines of code. The second stage achieves state-of-the-art performance on the recovery of empty categories by combining a linguistically-informed architecture and a rich feature set with the power of modern machine learning methods.

1 Introduction

The trees in the Penn Treebank (Bies et al., 1995) are annotated with a great deal of information to make various aspects of the predicate-argument structure easy to decode, including both function tags and markers of "empty" categories that represent displaced constituents. Modern statistical parsers such as (Collins, 2003) and (Charniak, 2000) however ignore much of this information and return only an impoverished version of the trees. While there has been some work in the last few years on enriching the output of state-of-the-art parsers that output Penn Treebank-style trees with function tags (e.g. (Blaheta, 2003)) or empty categories (e.g. (Johnson, 2002; Dienes and Dubey, 2003a; Dienes and Dubey, 2003b), only one system currently available, the dependency graph parser of (Jijkoun and de Rijke, 2004), recovers some representation of both these aspects of the Treebank representation; its output, however, cannot be inverted to recover the original tree structures. We present here a parser,[1] the first we know of, that recovers full Penn Treebank-style trees. This parser uses a minimal modification of the Collins parser to recover function tags, and then uses this enriched output to achieve or better state-of-the-art performance on recovering empty categories.

We focus here on Treebank-style output for two reasons: First, annotators developing additional treebanks in new genres of English that conform to the Treebank II style book (Bies et al., 1995) must currently add these additional annotations by hand, a much more laborious process than correcting parser output (the currently used method for annotating the skeletal structure itself). Our new parser is now in use in a new Treebank annotation effort. Second, the accurate recovery of semantic structure from parser output requires establishing the equivalent of the information encoded within these representations.

Our parser consists of two components. The first-stage is a modification of Bikel's implementation (Bikel, 2004) of Collins' Model 2 that recovers function tags while parsing. Remarkably little modification to the parser is needed to allow it to produce function tags as part of its output, yet without decreasing the regular Parseval metric. While it is difficult to evaluate function tag assignment in isola-

We would like to thank Fernando Pereira, Dan Bikel, Tony Kroch and Mark Liberman for helpful suggestions. This work was supported in part under the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-C-0022, and in part by the National Science Foundation under grants NSF IIS-0520798 and NSF EIA 02-05448 and under an NSF graduate fellowship.

[1] The parser consists of two boxes; those who prefer to label it by its structure, as opposed to what it does, might call it a *parsing system*.

Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL, pages 184–191,
New York, June 2006.

```
(S
  (NP-SBJ (DT The) (NN luxury)
          (NN auto) (NN maker) )
  (NP-TMP (JJ last) (NN year) )
  (VP (VBD sold)
    (NP (CD 1,214) (NNS cars) )
    (PP-LOC (IN in)
      (NP (DT the) (NNP U.S.) ))))
```

Figure 1: Example Tree

tion across the output of different parsers, our results match or exceed all but the very best of earlier tagging results, even though this earlier work is far more complicated than ours. The second stage uses a cascade of statistical classifiers which recovers the most important empty categories in the Penn Treebank style. These classifiers utilize a wide range of features, including crucially the function tags recovered in the first stage of parsing.

2 Motivation

Function tags are used in the current Penn Treebanks to augment nonterminal labels for various syntactic and semantic roles (Bies et al., 1995). For example, in Figure 1, `-SBJ` indicates the subject, `-TMP` indicates that the `NP last year` is serving as a temporal modifier, and `-LOC` indicates that the `PP` is specifying a location. Note that without these tags, it is very difficult to determine which of the two NPs directly dominated by S is in fact the subject. There are twenty function tags in the Penn Treebank, and following (Blaheta, 2003), we collect them into the five groups shown in Figure 2. While nonterminals can be assigned tags from different groups, they do not receive more than one tag from within a group. The Syntactic and Semantic groups are by far the most common tags, together making up over 90% of the function tag instances in the Penn Treebank.

Certain non–local dependencies must also be included in a syntactic analysis if it is to be most useful for recovering the predicate–argument structure of a complex sentence. For instance, in the sentence "The dragon I am trying to slay is green," it is important to know that *I* is the semantic subject and *the dragon* the semantic object of the slaying. The Penn Treebank (Bies et al., 1995) represents such dependencies by including nodes with no overt content (empty categories) in parse trees. In this work, we consider the three most frequent[2] and semantically important types of empty category annotations in most Treebank genres:

Null complementizers are denoted by the symbol `0`. They typically appear in places where, for example, an optional *that* or *who* is missing: "The king said `0` he could go." or "The man `(0)` I saw."

Traces of *wh*–movement are denoted by `*T*`, such as the noun phrase trace in "What$_1$ do you want `(NP *T*-1)`?" Note that *wh*–traces are co–indexed with their antecedents.

`(NP *)`s are used for several purposes in the Penn Treebank. Among the most common are passivization "`(NP-1` I) was captured `(NP *-1)`," and control "`(NP-1` I) tried `(NP *-1)` to get the best results."

Under this representation the above sentence would look like "(NP-1 The dragon) 0 (NP-2 I) am trying (NP *-2) to slay (NP *T*-1) is green."

Despite their importance, these annotations have largely been ignored in statistical parsing work. The importance of returning this information for most real applications of parsing has been greatly obscured by the Parseval metric (Black et al., 1991), which explicitly ignores both function tags and null elements. Because much statistical parsing research has been driven until recently by this metric, which has never been updated, the crucial role of parsing in recovering semantic structure has been generally ignored. An early exception to this was (Collins, 1997) itself, where Model 2 used function tags during the training process for heuristics to identify arguments (e.g., the `TMP` tag on the `NP` in Figure 1 disqualifies the `NP-TMP` from being treated as an argument). However, after this use, the tags are ignored, not included in the models, and absent from the parser output. Collins' Model 3 attempts to recover traces of *W*h-movement, with limited success.

3 Function Tags: Approach

Our system for restoring function tags is a modification of Collins' Model 2. We use the (Bikel, 2004)

[2]Excepting empty units (e.g. "$ 1,000,000 *U*"), which are not very interesting. According to Johnson, `(NP *)`s occur 28,146 times in the training portion of the PTB, `(NP *T*)`s occur 8,620 times, `0`s occur 7,969 times, and `(ADVP *T*)`s occur 2,492 times. In total, the types we consider cover roughly 84% of all the instances of empty categories in the training corpus.

Syntactic (55.9%)		Semantic (36.4%)				Misc (1.0%)		CLR (5.8%)	
DTV	Dative	NOM	Nominal	EXT	Extent	CLF	It-cleft	CLR	Closely-Related
LGS	Logical subj	ADV	Non-specific	LOC	Location	HLN	Headline		
PRD	Predicate		Adverbial	MNR	Manner	TTL	Title		
PUT	LOC of 'put'	BNF	Benefactive	PRP	Purpose			**Topicalization (2.6%)**	
SBJ	Subject	DIR	Direction	TMP	Temporal				
VOC	Vocative							TPC	Topic

Figure 2: Function Tags - Also shown is the percentage of each category in the Penn Treebank

emulation of the Collins parser.[3] Remarkably little modification to the parser is needed to allow it to produce function tags as part its output, without decreasing the regular Parseval metric.

The training process for the unmodified Collins parser carries out various preprocessing steps, which modify the trees in various ways before taking observations from them for the model. One of these steps is to identify and mark arguments with a parser internal tag (`-A`), using function tags as part of the heuristics for doing so. A following preprocessing step then deletes the original function tags.

We modify the Collins parser in a very simple way: the parser now retains the function tags after using them for argument identification, and so includes them in all the parameter classes. We also augment the argument identification heuristic to treat any nonterminal with any of the tags in the Syntactic group to be an argument; these are treated as synonyms for the internal tag that the parser uses to mark arguments. This therefore extends (Collins, 2003)'s use of function tags for excluding potential argument to also use them for including arguments.[4] The parser is then trained as before.

4 Function Tags: Evaluation

We compare our tagging results in isolation with the tagging systems of (Blaheta, 2003), since that work has the first highly detailed accounting of function tag results on the Penn Treebank, and with two recent tagging systems. We use both Blaheta's metric and his function tag groupings, shown in Figure 2, although our assignments are made by a fully integrated system. There are two aspects of Blaheta's metric that require discussion: First, this metric includes only constituents that were otherwise parsed correctly (ignoring function tag). Second, the metric ignores cases in which both the gold and test nonterminals are lacking function tags, since they would inflate the results.

5 Function Tags: Results

We trained the Bikel emulations of Collins' model 2 and our modified versions on sections 2-21 and tested on section 23. Scores are for all sentences, not just those with less than 40 words.

Parseval labelled recall/precision scores for the unmodified and modified parsers, show that there is almost no difference in the scores:

Parser	LR/LP
Model 2	88.12/88.31
Model 2-FuncB	88.23/88.31

We find this somewhat surprising, as we had expected that sparse data problems would arise, due to the shattering of `NP` into `NP-TMP`, `NP-SBJ`, etc.

Table 1 shows the overall results and the breakdown for the different function tag groups. For purposes of comparison, we have calculated our overall score both with and without `CLR`.[5] The (Blaheta, 2003) numbers in parentheses in Table 1 are from his feature trees specialized for the Syntactic and Semantic groups, while all his other numbers, including the overall score, are from using a single feature set for his four function tag groups.[6]

[3] Publicly available at `http://www.cis.upenn.edu/~dbikel/software.html`.

[4] Bikel's parser, in its latest version, already does something like this for Chinese and Arabic. However, the interaction with the subcat frame is different, in that it puts all nonterminals with a function tag into the miscellaneous slot in the subcat frame.

[5] (Jijkoun and de Rijke, 2004) do not state whether they are including `CLR`, but since they are comparing their results to (Blaheta and Charniak, 2000), we are assuming that they do. They do not break their results down by group.

[6] The P/R/F scores in (Blaheta, 2003)[p. 23] are internally

	— Overall —		— Breakdown by Function Tag Group —				
	w/CLR	**w/o CLR**	Syn	Sem	Top	Misc	CLR
Tag Group Frequency			*55.87%*	*36.40%*	*2.60%*	*1.03%*	*5.76%*
Model2-Ftags	88.95	90.78	95.76	84.56	93.89	17.31	65.86
Blaheta, 2003		88.28	95.16 (95.89)	79.81 (83.37)	93.72	39.44	
Jijkoun and de Rijke, 2004	88.50						
Musillo and Merlo, 2005			96.5	85.6			

Table 1: Overall Results (F-measure) and Breakdown by Function Tag Groups

Even though our tagging system results from only eliminating a few lines of code from the Collins parse, it has a higher overall score than (Blaheta, 2003), and a large increase over Blaheta's non-specialized Semantic score (79.81). It also outperforms even Blaheta's specialized Semantic score (83.37), and is very close to Blaheta's specialized score for the Syntactic group (95.89). However, since the evaluation is over a different set of non-terminals, arising from the different parsers,[7] it is difficult to draw conclusions as to which system is definitively "better". It does seem clear, though, that by integrating the function tags into the lexicalized parser, the results are roughly comparable with the post-processing work, and it is much simpler, without the need for a separate post-processing level or for specialized feature trees for the different tag groups.[8]

Our results clarify, we believe, the recent results of (Musillo and Merlo, 2005), now state-of-the-art, which extends the parser of

report a significant modification of the Henderson parser to incorporate strong notions of linguistic locality. They also manually restructure some of the function tags using tree transformations, and then train on these relabelled trees. Our results indicate that perhaps the simplest possible modification of an existing parser suffices to perform better than post-processing approaches. The linguistic sophistication of the work of (Musillo and Merlo, 2005) then provides an added boost in performance over simple integration.

6 Empty Categories: Approach

Most learning–based, phrase–structure–based (PSLB) work[9] on recovering empty categories has fallen into two classes: those which integrate empty category recovery into the parser (Dienes and Dubey, 2003a; Dienes and Dubey, 2003b) and those which recover empty categories from parser output in a post–processing step (Johnson, 2002; Levy and Manning, 2004). Levy and Manning note that thus far no PSLB post–processing approach has come close to matching the integrated approach on the most numerous types of empty categories.

However, there is a rule–based post–processing approach consisting of a set of entirely hand–designed rules (Campbell, 2004) which has better

inconsistent for the Semantic and Overall scores. We have kept the Precision and Recall and recalculated the F-measures, adjusting the Semantic score upwards from 79.15% to 79.81% and the Overall score downward from 88.63% to 88.28%.

[7] And the (Charniak, 2000) parser that (Blaheta, 2003) used has a reported F-measure of 89.5, higher than the Bikel parser used here.

[8] Our score on the Miscellaneous category is significantly lower, but as can be seen from Figure 2 and repeated in 1, this is a very rare category.

[9] As above, we consider only that work which both inputs and outputs phrase–structure trees. This notably excludes Jijkoun and de Rijke (Jijkoun and de Rijke, 2004), who have a system which seems to match the performance of Dienes and Dubey. However, they provide only aggregate statistics over all the types of empty categories, making any sort of detailed comparison impossible. Finally, it is not clear that their numbers are in fact comparable to those of Dienes and Dubey on parsed data because the metrics used are not quite equivalent, particularly for `(NP *)`s: among other differences, unlike Jijkoun and de Rijke's metric (taken from (Johnson, 2002)), Dienes and Dubey's is sensitive to the string extent of the antecedent node, penalizing them if the parser makes attachment errors involving the antecedent even if the system recovered the long–distance dependency itself correctly. Johnson noted that the two metrics did not seem to differ much for his system, but we found that evaluating our system with the laxer metric reduced error by 20% on the crucial task of restoring and finding the antecedents of `(NP *)`s, which make up almost half the empty categories in the Treebank.

results than the integrated approach. Campbell's rules make heavy use of aspects of linguistic representation unexploited by PSLB post–processing approaches, most importantly function tags and argument annotation.[10]

7 Empty Categories: Method

7.1 Runtime

The algorithm applies a series five maximum–entropy and two perceptron–based classifiers:

[1] For each PP, VP, and S node, ask the classifier NPTRACE to determine whether to insert an (NP *) as the object of a preposition, an argument of a verb, or the subject of a clause, respectively.

[2] For each node , ask NULLCOMP to determine whether or not to insert a 0 to the right.

[3] For each S node , ask WHXPINSERT to determine whether or not to insert a null *wh*–word to the left. If one should be, ask WHXPDISCERN to decide if it should be a (WHNP 0) or a (WHADVP 0).

[4] For each S which is a sister of WHNP or WHADVP, consider all possible places beneath it a *wh*–trace could be placed. Score each of them using WHTRACE, and insert a trace in the highest scoring position.

[5] For any S lacking a subject, insert (NP *).

[6] For each (NP *) in subject position, look at all NPs which c–command it. Score each of these using PROANTECEDENT, and co–index the (NP *) with the NP with the highest score. For all (NP *)s in non–subject positions, we follow Campbell in assigning the local subject as the controller.

[7] For each (NP *), ask ANTECEDENTLESS to determine whether or not to remove the co–indexing between it and its antecedent.

The sequencing of classifiers and choice of how to frame the classification decisions closely follows Campbell with the exception of finding antecedents of (NP *)s and inserting *wh*–traces, which follow Levy and Manning in using a competition–based approach. We differ from Levy and Manning in using a perceptron–based approach for these, rather than a maximum–entropy one. Also, rather than introducing an extra zero node for uncontrolled (NP *)s, we always assign a controller and then remove co–indexing from uncontrolled (NP *)s using a separate classifier.

[10]The non–PSLB system of Jijkoun and de Rijke uses function tags, and Levy and Manning mention that the lack of this information was sometimes an obstacle for them. Also, access to argument annotation inside the parser may account for a part of the good performance of Dienes and Dubey.

7.2 Training

Each of the maximum–entropy classifiers mentioned above was trained using MALLET (McCallum, 2002) over a common feature set. The most notable departure of this feature list from previous ones is in the use of function tags and argument markings, which were previously ignored for the understandable reason that though they are present in the Penn Treebank, parsers generally do not produce them. Another somewhat unusual feature examined right and left sisters.

The PROANTECEDENT perceptron classifier uses the local features of the controller and the controlled (NP *), whether the controller precedes or follows the controlled (NP *), the sequence of categories on the path between the two (with the 'turning' category marked), the length of that path, and which categories are contained anywhere along the path.

The WHTRACE perceptron classifier uses the following features each conjoined with the type of *wh*–trace being sought: the sequence of categories found on the path between the trace and its antecedent, the path length, which categories are contained anywhere along the path, the number of bounding categories crossed and whether the trace placement violates subjacency, whether or not the trace insertion site's parent is the first verb on the path, whether or not the insertion site's parent contains another verb beneath it, and if the insertion site's parent is a verb, whether or not the verb is saturated.[11]

All maximum–entropy classifiers were trained on sections 2-21 of the Penn Treebank's Wall Street Journal section; the perceptron–based classifiers were trained on sections 10-18. Section 24 was used for development testing while choosing the feature

[11]To provide the verb saturation feature, we calculated the number of times each verb in the training corpus occurs with each number of NP arguments (both overt and traces). When calculating the feature value, we compare the number of instances seen in the training corpus of the verb with the number of argument NPs it overtly has with the number of times in the corpus the verb occurs with one more argument NP.

set and other aspects of the system, and section 23 was used for the final evaluation.

8 Empty Categories: Results

8.1 Metrics

For the sake of easy comparison, we report our results using the most widely–used metric for performance on this task, that proposed by Johnson. This metric judges an entity correct if it matches the gold standard in type and string position (and, if there is an antecedent, in its label and string extent). Because Campell reports results by category using only his own metric, we use this metric to compare our results to his. There is much discussion in the literature of metrics for this task; Levy and Manning and Campbell both note that the Johnson metric fails to catch when an empty category has a correct string position but incorrect parse tree attachment. While we do not have space to discuss this issue here, the metrics they in turn propose also have significant weaknesses. In any event, we use the metrics that allow the most widespread comparison.

8.2 Comparison to Other PSLB Methods

Category	Pres	LM	J	DD
Comb. 0	**87.8**	87.0	77.1	
COMP-SBAR	**91.9**		88.0	85.5
COMP-WHNP	**61.5**		47.0	48.8
COMP-WHADVP	**69.0**			
NP *	69.1	61.1	55.6	**70.3**
Comb. *wh*–trace	**78.2**	63.3	75.2	75.3
NP *T*	80.9		80.0	**82.0**
ADVP *T*	**69.8**		56	53.6

Table 2: F1 scores comparing our system to the two PSLB post–processing systems and Dienes and Dubey's integrated system on automatically parsed trees from section 23 using Johnson's metric.

F1 scores on parsed sentences from section 23 are given in table 2. Note that our system's parsed scores were obtained using our modified version of Bikel's implementation of Collins's thesis parser which assigns function tags, while the other PSLB post–processing systems use Charniak's parser (Charniak, 2000) and Dienes and Dubey integrate empty category recovery directly into a variant of Collins's parser.

On parsed trees, our system outperforms other PSLB post–processing systems. On the most numerous category by far, `(NP *)`, our system reduces the error of the best PSLB post–processing approach by 21%. Comparing our aggregate *wh*–trace results to the others,[12] we reduce error by 41% over Levy and Manning and by 12% over Johnson.

System	Precision	Recall	F1
D&D	**78.50**	68.08	72.92
Pres	74.70	**74.62**	**74.66**

Table 3: Comparison of our system with that of Dienes and Dubey on parsed data from section 23 over the aggregation of all categories in table 2 excepting the infrequent `(WHADVP 0)`s, which they do not report but which we almost certainly outperform them on.

Performance on parsed data compared to the integrated system of Dienes and Dubey is split. We reduce error by 25% and 44% on plain `0`s and `(WHNP 0)`s, respectively and by 12% on *wh*–traces. We increase error by 4% on `(NP *)`s. Aggregating over all the categories under consideration, the more balanced precision and recall of our system puts it ahead of Dienes and Dubey's, with a 6.4% decrease in error (table 3).

8.3 Comparison to Campbell

Category	Present	Campbell
NP *	**88.8**	86.9
NP *T*	**96.3**	96.0
ADVP *T*	**82.2**	79.9
0	**99.8**	98.5

Table 4: A comparison of the present system with Campbell's rule–based system on gold–standard trees from section 23 using Campbell's metric

[12] Levy and Manning report Johnson to have an aggregate *wh*–trace score of 80, but Johnson's paper gives 80 as his score for `(NP *T*)`s only, with 56 as his score for `(ADVP *T*)`s. A similar problem seems to have occured with Levy and Manning's numbers for Dienes and Dubey on this and on `(NP *)`s. This error makes the other two systems appear to outperform Levy and Manning on *wh*–traces by a slightly larger margin than they actually do.

Classifier	Features with largest weights
NPTRACE	daughter categories, function tags, argumentness, heads, and POS tags, subjectless S...
NULLCOMP	is first daughter?, terminalness, aunt's label and POS tag, mother's head, daughters' heads, great–grandmother's label...
WHXPINSERT	is first daughter?, left sister's terminalness, labels of mother, aunt, and left sister, aunt's head...
WHXPDISCERN	words contained by grandmother, grandmother's head, aunt's head, grandmother's function tags, aunt's label, aunt's function tags...
WHTRACE	lack of subject, daughter categories, child argument information, subjacency violation, saturation, whether or not there is a verb below, path information...
PROANTECEDENT	controller's sisters' function tags, categories path contains, path length, path shape, controller's function tags, controller's sisters' heads, linear precedence information...
ANTECEDENTLESS	mother's function tags, great–grandmother's label, aunt's head ("It is difficult to..."), grandmother's function tag, mother's head...

Table 5: A few of the most highly weighted features for various classifiers

On gold-standard trees,[13] our system outperforms Campbell's rule–based system on all four categories, reducing error by 87% on `0`s,[14] by 11% on `(ADVP *T*)`s, by 7% on `(NP *T*)`s, and by 8% on the extremely numerous `(NP *)`s.

9 Empty Categories: Discussion

We have shown that a PSLB post–processing approach can outperform the state–of–the–art integrated approach of Dienes and Dubey.[15] Given that their modifications to Collins's parser caused a decrease in local phrase structure parsing accuracy due to sparse data difficulties (Dienes and Dubey, 2003a), our post–processing approach seems to be an especially attractive choice. We have further shown that our PSLB approach, using only simple, unconjoined features, outperforms Campbell's state–of–the–art, complex system on gold–standard data, suggesting that much of the power of his system lies in his richer linguistic representation and his structuring of decisions rather than the hand–designed rules.

We have also compared our system to that of Levy and Manning which is based on a similar learning technique and have shown large increases in performance on all of the most common types of empty categories; this increase seems to have come almost entirely from an enrichment of the linguistic representation and a slightly different structuring of the problem, rather than any use of more powerful machine–learning techniques

We speculate that the primary source of our performance increase is the enrichment of the linguistic representation with function tags and argument markings from the parser's first stage, as table 5 attests. We also note that several classifiers make use of the properties of aunt nodes, which have previously been exploited only in a limite form in Johnson's patterns. For example, ANTECEDENTLESS uses the aunt's head word to learn an entire class of uncontrolled PRO constructions like "It is difficult `(NP *)` to imagine living on Mars."

10 Conclusion

This work has presented a two stage parser that recovers Penn Treebank style syntactic analyses of new sentences including skeletal syntactic structure, and, for the first time, both function tags and empty categories. The accuracy of the first-stage parser on the standard Parseval metric matches that of the (Collins, 2003) parser on which it is based, despite the data fragmentation caused by the greatly enriched space of possible node labels for the Collins statistical model. This first stage simultaneously achieves near state-of-the-art performance on recovering function tags with minimal modifications to the underlying parser, modifying less than ten lines

[13] Only aggregate statistics over a different set of empty categories were available for Campbell on parsed data, making a comparison impossible.

[14] Note that for comparison with Campbell, the `0` numbers here exclude `(WHNP 0)`s and `(WHADVP 0)`s.

[15] And therefore also very likely outperforms the dependency–based post–processing approach of Jijkoun and de Rijke, even if its performance does in fact equal Dienes and Dubey's.

of code. We speculate that this success is due to the lexicalization of the Collins model, combined with the sophisticated backoff structure already built into the Collins model. The second stage achieves state-of-the-art performance on the recovery of empty categories by combining the linguistically-informed architecture of (Campbell, 2004) and a rich feature set with the power of modern machine learning methods. This work provides an example of how small enrichments in linguistic representation and changes in the structure of the problem having significant effects on the performance of a machine–learning–based system. More concretely, we showed for the first time that a PSLB post–processing system can outperform the state–of–the–art for both rule–based post–processing and integrated approaches to the empty category restoration problem.

Most importantly from the point of view of the authors, we have constructed a system that recovers sufficiently rich syntactic structure based on the Penn Treebank to provide rich syntactic guidance for the recovery of predicate-argument structure in the near future. We also expect that productivity of syntactic annotation of further genres of English will be significantly enhanced by the use of this new tool, and hope to have practical evidence of this in the near future.

References

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for Treebank II style Penn Treebank project. Technical report, University of Pennsylvania.

Daniel M. Bikel. 2004. *On the Parameter Space of Lexicalized Statistical Parsing Models*. Ph.D. thesis, Department of Computer and Information Sciences, University of Pennsylvania.

E. Black, S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Fourth DARPA Workshop on Speech and Natural Language*, pages 306–311, CA.

Don Blaheta and Eugene Charniak. 2000. Assigning function tags to parsed text. In *Proceedings of the 1st Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 234–240, Seattle.

Don Blaheta. 2003. *Function Tagging*. Ph.D. thesis, Brown University.

Richard Campbell. 2004. Using linguistic principles to recover empty categories. In *Proceedings of ACL*.

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

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, Madrid.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29:589–637.

Peter Dienes and Amit Dubey. 2003a. Antecedent recovery: Experiments with a trace tagger. In *Proceedings of EMNLP*.

Peter Dienes and Amit Dubey. 2003b. Deep processing by combining shallow methods. In *Proceedings of ACL*.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of NLT-NAACL 2003*, Edmonton, Alberta, Canada. Association for Computational Linguistics.

Valentin Jijkoun and Maarten de Rijke. 2004. Enriching the output of a parser using memory-based learning. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, Barcelona, Spain.

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

Roger Levy and Christopher Manning. 2004. Deep dependencies from context–free statistical parsers: correcting the surface dependency approximation. In *Proceedings of the ACL*.

Andrew Kachites McCallum. 2002. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

Gabriele Musillo and Paolo Merlo. 2005. Lexical and structural biases for function parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 83–92, Vancouver, British Columbia, October. Association for Computational Linguistics.