Recovering latent information in treebanks

David Chiang and **Daniel M. Bikel**
University of Pennsylvania
Dept of Computer and Information Science
200 S 33rd Street
Philadelphia PA 19104 USA
{dchiang,dbikel}@cis.upenn.edu

Abstract

Many recent statistical parsers rely on a preprocessing step which uses hand-written, corpus-specific rules to augment the training data with extra information. For example, head-finding rules are used to augment node labels with lexical heads. In this paper, we provide machinery to reduce the amount of human effort needed to adapt existing models to new corpora: first, we propose a flexible notation for specifying these rules that would allow them to be shared by different models; second, we report on an experiment to see whether we can use Expectation-Maximization to automatically fine-tune a set of hand-written rules to a particular corpus.

1 Introduction

Most work in statistical parsing does not operate in the realm of parse trees as they appear in many treebanks, but rather on trees transformed via augmentation of their node labels, or some other transformation (Johnson, 1998). This methodology is illustrated in Figure 1. The information included in the node labels' augmentations may include lexical items, or a node label suffix to indicate the node is an argument and not an adjunct; such extra information may be viewed as latent information, in that it is not directly present in the treebank parse trees, but may be recovered by some means. The process of recovering this latent information has largely been limited to the hand-construction of heuristics. However, as is often the case, hand-constructed heuristics may not be optimal or very robust. Also, the effort required to construct such rules can be considerable. In both respects, the use of such rules runs counter to the data-driven approach to statistical parsing.

In this paper, we propose two steps to address this problem. First, we define a new, fairly simple syntax for the identification and transformation of node labels that accommodates a wide variety of node-label augmentations, including all those that are performed by existing statistical parsers that we have examined. Second, we explore a novel use of Expectation-Maximization (Dempster et al., 1977) that iteratively reestimates a parsing model using the augmenting heuristics as a starting point. Specifically, the EM algorithm we use is a variant of the Inside-Outside algorithm (Baker, 1979; Lari and Young, 1990; Hwa, 1998). The reestimation adjusts the model's parameters in the augmented parse-tree space to maximize the likelihood of the observed (incomplete) data, in the hopes of finding a better distribution over augmented parse trees (the complete data). The ultimate goal of this work is to minimize the human effort needed when adapting a parsing model to a new domain.

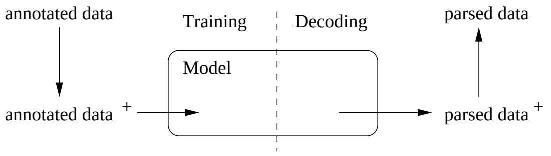

Figure 1: Methodology for the development of a statistical parser. A + indicates augmentation.

2 Background

2.1 Head-lexicalization

Many of the recent, successful statistical parsers have made use of lexical information or an implicit lexicalized grammar, both for English and, more recently, for other languages. All of these parsers recover the "hidden" lexicalizations in a treebank and find the most probable lexicalized tree when parsing, only to strip out this hidden information prior to evaluation. Also, in all these parsing efforts lexicalization has meant finding heads of constituents and then propagating those lexical heads to their respective parents. In fact, nearly identical head-lexicalizations were used in the dis-

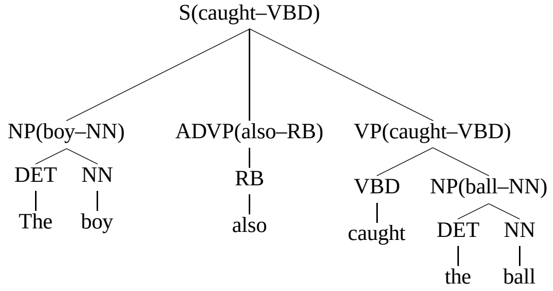

Figure 2: A simple lexicalized parse tree.

criminative models described in (Magerman, 1995; Ratnaparkhi, 1997), the lexicalized PCFG models in (Collins, 1999), the generative model in (Charniak, 2000), the lexicalized TAG extractor in (Xia, 1999) and the stochastic lexicalized TAG models in (Chiang, 2000; Sarkar, 2001; Chen and Vijay-Shanker, 2000). Inducing a lexicalized structure based on heads has a two-pronged effect: it not only allows statistical parsers to be sensitive to lexical information by including this information in the probability model's dependencies, but it also determines which of all possible dependencies—both syntactic and lexical—will be included in the model itself. For example, in Figure 2, the nonterminal NP(boy–NN) is dependent on VP(caught–VBD) and not the other way around.

2.2 Other tree transformations

Lexicalization via head-finding is but one of many possible tree transformations that might be useful for parsing. As explored thoroughly by Johnson (1998), even simple, local syntactic transformations on training trees for an unlexicalized PCFG model can have a significant impact on parsing performance. Having picked up on this idea, Collins (1999) devises rules to identify arguments, *i.e.*, constituents that are required to exist on a particular side of a head child constituent dominated by a particular parent. The parsing model can then probabilistically predict sets of requirements on either side of a head constituent, thereby incorporating a type of subcategorization information. While the model is augmented to include this subcat-prediction feature, the actual identification of arguments is performed as one of many preprocessing steps on training trees, using a set of rules similar to those used for the identification of heads. Also, (Collins, 1999) makes use of several other transformations, such as the identification of subjectless sentences (augmenting *S* nodes to become *SG*) and the augmentation of nonterminals for gap threading. Xia (1999) combines head-finding with argument identification to extract elementary trees for use in the lexicalized TAG formalism. Other researchers investigated this type of extraction to construct stochastic TAG parsers (Chiang, 2000; Chen and Vijay-Shanker, 2000; Sarkar, 2001).

2.3 Problems with heuristics

While head-lexicalization and other tree transformations allow the construction of parsing models with more data-sensitivity and richer representations, crafting rules for these transformations has been largely an art, with heuristics handed down from researcher to researcher. What's more, on top of the large undertaking of designing and implementing a statistical parsing model, the use of heuristics has required a further effort, forcing the researcher to bring both linguistic intuition and, more often, engineering savvy to bear whenever moving to a new treebank. For example, in the rule sets used by the parsers described in (Magerman, 1995; Ratnaparkhi, 1997; Collins, 1999), the sets of rules for finding the heads of *ADJP*, *ADVP*, *NAC*, *PP* and *WHPP* include rules for picking either the rightmost or leftmost *FW* (foreign word). The apparently haphazard placement of these rules that pick out *FW* and the rarity of *FW* nodes in the data strongly suggest these rules are the result of engineering effort. Furthermore, it is not at all apparent that tree-transforming heuristics that are useful for one parsing model will be useful for another. Finally, as is often the case with heuristics, those used in statistical parsers tend not to be data-sensitive, and ironically do not rely on the words themselves.

3 Rule-based augmentation

In the interest of reducing the effort required to construct augmentation heuristics, we would like a notation for specifying rules for selecting nodes in bracketed data that is both flexible enough to encode the kinds of rule sets used by existing parsers, and intuitive enough that a rule set for a new language can be written easily without knowledge of computer programming. Such a notation would simplify the task of writing new rule sets, and facilitate experimentation with different rules. Moreover, rules written in this notation would be interchangeable between different models, so that, ideally, adaptation of a model to a new corpus would be trivial.

We define our notation in two parts: a *structure pattern language*, whose basic patterns are specifications of single nodes written in a *label pattern language*.

3.1 Structure patterns

Most existing head-finding rules and argument-finding rules work by specifying parent-child relations (*e.g.*, **NN** is the head of **NP**, or **NP** is an argument of **VP**). A generalization of this scheme that is familiar to linguists and computer scientists alike would be a context-free grammar with rules of the form

$$A \rightarrow A_1 \cdots (A_i)^l \cdots A_n,$$

where the superscript l specifies that if this rule gets used, the ith child of A should be marked with the label l.

However, there are two problems with such an approach. First, writing down such a grammar would be tedious to say the least, and impossible if we want to handle trees with arbitrary branching factors. So we can use an *extended CFG* (Thatcher, 1967), a CFG whose right-hand sides are regular expressions. Thus we introduce a union operator ($\cup$) and a Kleene star ($*$) into the syntax for right-hand sides.

The second problem that our grammar may be ambiguous. For example, the grammar

$$\mathbf{X} \rightarrow \mathbf{Y^hY} \cup \mathbf{YY^h}$$

could mark with an **h** either the first or second symbol of **YY**. So we impose an ordering on the rules of the grammar: if two rules match, the first one wins. In addition, we make the $\cup$ operator noncommutative: $\alpha \cup \beta$ tries to match α first, and β only if it does not match α, as in Perl. (Thus the above grammar would mark the first **Y**.) Similarly, α^* tries to match as many times as possible, also as in Perl.

But this creates a third and final problem: in the grammar

$$\mathbf{X} \rightarrow (\mathbf{YY^h} \cup \mathbf{Y^h})(\mathbf{YY} \cup \mathbf{Y}),$$

it is not defined which symbol of **YYY** should be marked, that is, which union operator takes priority over the other. Perl circumvents this problem by always giving priority to the left. In algebraic terms, concatenation left-distributes over union but does not right-distribute over union in general.

However, our solution is to provide a pair of concatenation operators: $\succ$, which gives priority to the left, and $\prec$, which gives priority to the right:

$$\mathbf{X} \rightarrow (\mathbf{YY^h} \cup \mathbf{Y^h}) \succ (\mathbf{YY} \cup \mathbf{Y}) \quad (1)$$

$$\mathbf{X} \rightarrow (\mathbf{YY^h} \cup \mathbf{Y^h}) \prec (\mathbf{YY} \cup \mathbf{Y}) \quad (2)$$

Rule (1) marks the second **Y** in **YYY**, but rule (2) marks the first **Y**. More formally,

$$\begin{aligned} \alpha \prec (\beta \cup \gamma) &= (\alpha \prec \beta) \cup (\alpha \prec \gamma) \\ (\beta \cup \gamma) \succ \alpha &= (\beta \succ \alpha) \cup (\gamma \succ \alpha) \end{aligned}$$

But if α contains no unions or Kleene stars, then

$$\begin{aligned} \alpha \succ \beta &= \alpha \prec \beta \qquad (\equiv \alpha\beta) \\ \beta \succ \alpha &= \beta \prec \alpha \qquad (\equiv \beta\alpha) \end{aligned}$$

So then, consider the following rules:

$$\mathbf{VP} \rightarrow \bot^* \succ \mathbf{VB^h} \succ \bot^*, \quad (3)$$

$$\mathbf{VP} \rightarrow \bot^* \prec \mathbf{VB^h} \prec \bot^*. \quad (4)$$

where $\bot$ is a wildcard pattern which matches any single label (see below). Rule (3) mark with an **h** the rightmost **VB** child of a **VP**, whereas rule (4) marks the leftmost **VB**. This is because the Kleene star always prefers to match as many times as possible, but in rule (3) the first Kleene star's preference takes priority over the last's, whereas in rule (4) the last Kleene star's preference takes priority over the first's.

Consider the slightly more complicated examples:

$$\mathbf{VP} \rightarrow \bot^* \prec (\mathbf{VB^h} \cup \mathbf{MD^h}) \prec \bot^* \quad (5)$$

$$\mathbf{VP} \rightarrow \bot^* \prec ((\mathbf{VB^h} \cup \mathbf{MD^h}) \succ \bot^*) \quad (6)$$

Rule (5) marks the leftmost child which is either a **VB** or a **MD**, whereas rule (6) marks the leftmost **VB** if any, or else the leftmost **MD**. To see why this so, consider the string **MD VB X**. Rule (5) would mark the **MD** as **h**, whereas rule (6) would mark the **VB**. In both rules **VB** is preferred over **MD**, and symbols to the left over symbols to the right, but in rule (5) the leftmost preference (that is, the preference of the last Kleene star to match as many times as possible) takes priority, whereas in rule (6) the preference for **VB** takes priority.

3.2 Label patterns

Since nearly all treebanks have complex nonterminal alphabets, we need a way of concisely specifying classes of labels. Unfortunately, this will necessarily vary somewhat across treebanks: all we can define that is truly treebank-independent is the $\bot$ pattern, which matches any label. For Penn Treebank II style annotation (Marcus et al., 1993), in which a nonterminal symbol is a category together with zero or more functional tags, we adopt the following scheme: the atomic pattern a matches any label with category a or functional tag a; moreover, we define Boolean operators $\wedge$, $\vee$, and $\neg$. Thus **NP** $\wedge$ $\neg$**ADV** matches **NP–SBJ** but not **NP–ADV**.[1]

3.3 Summary

Using the structure pattern language and the label pattern language together, one can fully encode the head/argument rules used by Xia (which resemble (5) above), and the family of rule sets used by Black, Magerman, Collins, Ratnaparkhi, and others (which resemble (6) above). In Collins' version of the head rules, **NP** and **PP** require special treatment, but these can be encoded in our notation as well.

4 Unsupervised learning of augmentations

In the type of approach we have been discussing so far, hand-written rules are used to augment the training data, and this augmented training data is then used to train a statistical model. However, if we train the model by maximum-likelihood estimation, the estimate we get will indeed maximize the likelihood of the training data as augmented by the hand-written rules, but not necessarily that of the training data itself. In this section we explore the possibility of training a model directly on unaugmented data.

A generative model that estimates $P(S, T, T^+)$ (where T^+ is an augmented tree) is normally used for parsing, by computing the most likely (T, T^+) for a given S. But we may also use it for augmenting trees, by computing the most likely T^+ for a given sentence-tree pair (S, T). From the latter perspective, because its trees are unaugmented, a treebank is a corpus of incomplete data, warranting the use of unsupervised learning methods to reestimate a model that includes hidden parameters. The approach we take below is to seed a parsing model using hand-written rules, and then use the Inside-Outside algorithm to reestimate its parameters. The resulting model, which locally maximizes the likelihood of the *unaugmented* training data, can then be used in two ways: one might hope that as a parser, it would parse more accurately than a model which only maximizes the likelihood of training data augmented by hand-written rules; and that as a tree-augmenter, it would augment trees in a more data-sensitive way than hand-written rules.

4.1 Background: tree adjoining grammar

The parsing model we use is based on the stochastic tree-insertion grammar (TIG) model described by Chiang (2000). TIG (Schabes and Waters, 1995) is a weakly-context free restriction of tree adjoining grammar (Joshi and Schabes, 1997), in which tree fragments called *elementary trees* are combined by two composition operations, *substitution* and *adjunction* (see Figure 3). In TIG there are certain restrictions on the adjunction operation. Chiang's model adds a third composition operation called *sister-adjunction* (see Figure 3), borrowed from D-tree substitution grammar (Rambow et al., 1995).[2]

There is an important distinction between *derived trees* and *derivation trees* (see Figure 3). A derivation tree records the operations that are used to combine elementary trees into a derived tree. Thus there is a many-to-one relationship between derivation trees and derived trees: every derivation tree specifies a derived tree, but a derived tree can be the result of several different derivations.

The model can be trained directly on TIG derivations if they are available, but corpora like the Penn Treebank have only derived trees. Just as Collins uses rules to identify heads and arguments and thereby lexicalize trees, Chiang uses nearly the same rules to reconstruct derivations: each training example is broken into elementary trees, with each head child remaining attached to its parent, each argument broken into a substitution node and an initial root, and each adjunct broken off as a modifier auxiliary tree.

However, in this experiment we view the derived trees in the Treebank as incomplete data, and try to reconstruct the derivations (the complete data) using the Inside-Outside algorithm.

4.2 Implementation

The expectation step (E-step) of the Inside-Outside algorithm is performed by a parser that computes all possible derivations for each parse tree in the training data. It then computes inside and outside probabilities as in Hwa's experiment (1998), and uses these to compute the expected number of times each event occurred. For the maximization step (M-step), we obtain a maximum-likelihood estimate of the parameters of the model using relative-frequency es-

[1] Note that unlike the noncommutative union operator $\cup$, the disjunction operator $\vee$ has no preference for its first argument.

[2] The parameters for sister-adjunction in the present model differ slightly from the original. In the original model, all the modifier auxiliary trees that sister-adjoined at a particular position were generated independently, except that each sister-adjunction was conditioned on whether it was the first at that position. In the present model, each sister-adjunction is conditioned on the root label of the previous modifier tree.

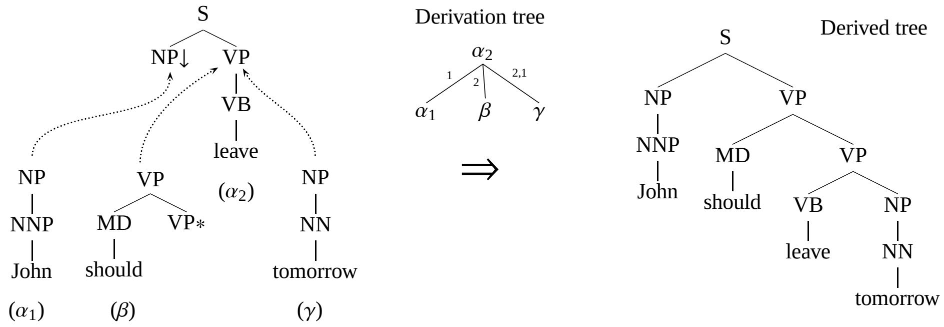

Figure 3: Grammar and derivation for "John should leave tomorrow." In this derivation, α_1 gets substituted, β gets adjoined, and γ gets sister-adjoined.

timation, just as in the original experiment, as if the expected values for the complete data were the training data.

Smoothing presents a special problem. There are several several backoff levels for each parameter class that are combined by deleted interpolation. Let ϕ_1, ϕ_2 and ϕ_3 be functions from full history contexts Y to less specific contexts at levels 1, 2 and 3, respectively, for some parameter class with three backoff levels (with level 1 using the most specific contexts). Smoothed estimates for parameters in this class are computed as follows:

$$e = \lambda_1 e_1 + (1 - \lambda_1)(\lambda_2 e_2 + (1 - \lambda_2) e_3)$$

where e_i is the estimate of $p(X \mid \phi_i(Y))$ for some future context X, and the λ_i are computed by the formula found in (Bikel et al., 1997), modified to use the multiplicative constant 5 found in the similar formula of (Collins, 1999):

$$\lambda_i = \left(1 - \frac{d_{i-1}}{d_i}\right)\left(\frac{1}{1 + 5u_i/d_i}\right) \tag{7}$$

where d_i is the number of occurrences in training of the context $\phi_i(Y)$ (and $d_0 = 0$), and u_i is the number of unique outcomes for that context seen in training.

There are several ways one might incorporate this smoothing into the reestimation process, and we chose to depart as little as possible from the original smoothing method: in the E-step, we use the smoothed model, and after the M-step, we use the original formula (7) to recompute the smoothing weights based on the new counts computed from the E-step. While simple, this approach has two important consequences. First, since the formula for the smoothing weights intentionally does *not* maximize the likelihood of the training data, each iteration of reestimation is not guaranteed to increase the likelihood of the training data. Second, reestimation tends to increase the size of the model in memory, since smoothing gives nonzero expected counts to many events which were unseen in training. Therefore, since the resulting model is quite large, if an event at a particular point in the derivation forest has an expected count below 10^{-15}, we throw it out.

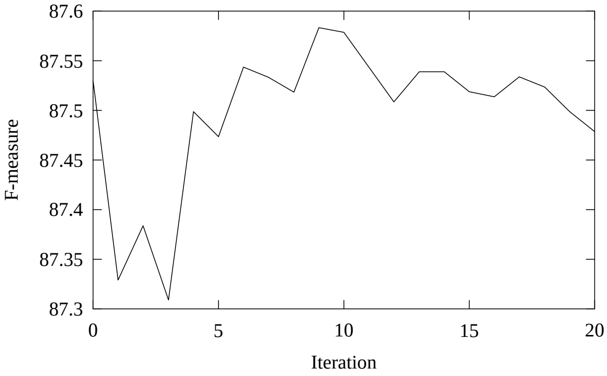

Figure 4: English, starting with full rule set

4.3 Experiment

We first trained the initial model on sections 02–21 of the WSJ corpus using the original head rules, and then ran the Inside-Outside algorithm on the same data. We tested each successive model on some held-out data (section 00), using a beam width of 10^{-4}, to determine at which iteration to stop. The F-measure (harmonic mean of labeled precision and recall) for sentences of length ≤ 100 for each iteration is shown in Figure 4. We then selected the ninth reestimated model and compared it with the initial model on section 23 (see Figure 7). This model did only marginally better than the initial model on section 00, but it actually performs worse than the initial model on section 23. One explanation is that the

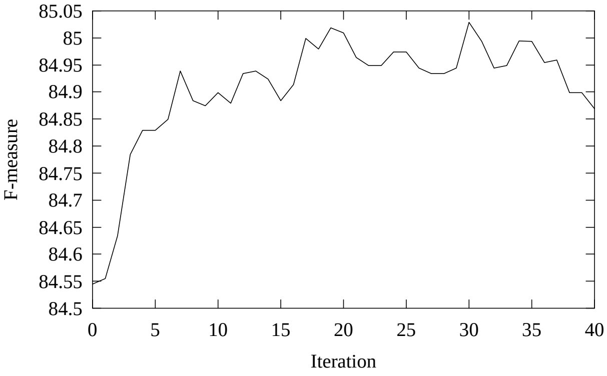

Figure 5: English, starting with simplified rule set

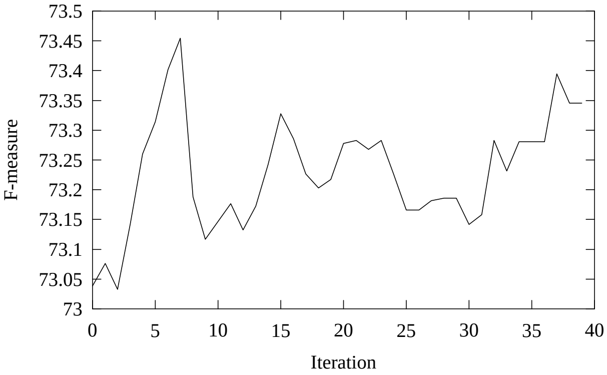

Figure 6: Chinese, starting with full rule set

head rules, since they have been extensively fine-tuned, do not leave much room for improvement. To test this, we ran two more experiments.

The second experiment started with a simplified rule set, which simply chooses either the leftmost or rightmost child of each node as the head, depending on the label of the parent: *e.g.*, for VP, the leftmost child is chosen; for NP, the rightmost child is chosen. The argument rules, however, were not changed. This rule set is supposed to represent the kind of rule set that someone with basic familiarity with English syntax might write down in a few minutes. The reestimated models seemed to improve on this simplified rule set when parsing section 00 (see Figure 5); however, when we compared the 30th reestimated model with the initial model on section 23 (see Figure 7), there was no improvement.

The third experiment was on the Chinese Treebank, starting with the same head rules used in (Bikel and Chiang, 2000). These rules were originally written by Xia for grammar development, and although we have modified them for parsing, they have not received as much fine-tuning as the English rules have. We trained the model on sections 001–270 of the Penn Chinese Treebank, and reestimated it on the same data, testing it at each iteration on sections 301–325 (Figure 6). We selected the 38th reestimated model for comparison with the initial model on sections 271–300 (Figure 7). Here we did observe a small improvement: an error reduction of 3.4% in the F-measure for sentences of length ≤ 40.

4.4 Discussion

Our hypothesis that reestimation does not improve on the original rule set for English because that rule set is already fine-tuned was partially borne out by the second and third experiments. The model trained with a simplified rule set for English showed improvement on held-out data during reestimation, but showed no improvement in the final evaluation; however, the model trained on Chinese did show a small improvement in both. We are uncertain as to why the gains observed during the second experiment were not reflected in the final evaluation, but based on the graph of Figure 5 and the results on Chinese, we believe that reestimation by EM can be used to facilitate adaptation of parsing models to new languages or corpora.

It is possible that our method for choosing smoothing weights at each iteration (see §4.2) is causing some interference. For future work, more careful methods should be explored. We would also like to experiment on the parsing model of Collins (1999), which, because it can recombine smaller structures and reorder subcategorization frames, might open up the search space for better reestimation.

5 Conclusion

Even though researchers designing and implementing statistical parsing models have worked in the methodology shown in Figure 1 for several years now, most of the work has focused on finding effective features for the model component of the methodology, and on finding effective statistical techniques for parameter estimation. However, there has been much behind-the-scenes work on the actual transformations, such as head finding, and most of this work has consisted of hand-tweaking existing heuristics. It is our hope that by introducing this new syntax, less toil will be needed to write nonterminal augmentation rules, and that human effort will be lessened further by the use of unsupervised methods such as the one presented here to produce better models for parsing and tree augmentation.

		≤ 100 words					≤ 40 words				
Model	Step	LR	LP	CB	0CB	≤ 2 CB	LR	LP	CB	0CB	≤ 2 CB
Original	initial	86.95	87.02	1.21	62.38	82.33	87.68	87.76	1.02	65.30	84.86
Original	9	86.37	86.71	1.26	61.42	81.79	87.18	87.48	1.06	64.41	84.23
Simple	initial	84.50	84.18	1.54	57.57	78.35	85.46	85.17	1.29	60.71	81.11
Simple	30	84.21	84.50	1.53	57.95	77.77	85.12	85.35	1.30	60.94	80.62
Chinese	initial	75.30	76.77	2.72	45.95	67.05	78.37	80.03	1.79	52.82	74.75
Chinese	38	75.20	77.99	2.66	47.69	67.63	78.79	81.06	1.69	54.15	75.08

Figure 7: Results on test sets. Original = trained on English with original rule set; Simple = English, simplified rule set. LR = labeled recall, LP = labeled precision; CB = average crossing brackets, 0 CB = no crossing brackets, ≤ 2 CB = two or fewer crossing brackets. All figures except CB are percentages.

Acknowledgments

This research was supported in part by NSF grant SBR-89-20230. We would like to thank Anoop Sarkar, Dan Gildea, Rebecca Hwa, Aravind Joshi, and Mitch Marcus for their valuable help.

References

James K. Baker. 1979. Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550.

Daniel M. Bikel and David Chiang. 2000. Two statistical parsing models applied to the Chinese Treebank. In *Proceedings of the Second Chinese Language Processing Workshop*, pages 1–6.

Daniel M. Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. 1997. Nymble: a high-performance learning name-finder. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP 1997)*, pages 194–201.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of ANLP-NAACL2000*, pages 132–139.

John Chen and K. Vijay-Shanker. 2000. Automated extraction of TAGs from the Penn Treebank. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 65–76, Trento.

David Chiang. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of ACL-2000*, pages 456–463.

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

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Stat. Soc. B*, 39:1–38.

Rebecca Hwa. 1998. An empirical evaluation of probabilistic lexicalized tree insertion grammars. In *Proceedings of COLING-ACL '98*, pages 557–563.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24:613–632.

Aravind K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In Grzegorz Rosenberg and Arto Salomaa, editors, *Handbook of Formal Languages and Automata*, volume 3, pages 69–124. Springer-Verlag, Heidelberg.

K. Lari and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

David M. Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of ACL '95*, pages 276–283.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330.

Owen Rambow, K. Vijay-Shanker, and David Weir. 1995. D-tree grammars. In *Proceedings of ACL '95*, pages 151–158.

Adwait Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP-2)*.

Anoop Sarkar. 2001. Applying co-training methods to statistical parsing. In *Proceedings of NAACL-2001*, pages 175–182.

Yves Schabes and Richard C. Waters. 1995. Tree insertion grammar: a cubic-time parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21:479–513.

J. W. Thatcher. 1967. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *J. Comp. Sys. Sci.*, 1:317–322.

Fei Xia. 1999. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, pages 398–403.