Fast LR Parsing Using Rich (Tree Adjoining) Grammars

Carlos A. Prolo
Department of Computer and Information Science
University of Pennsylvania
prolo@linc.cis.upenn.edu

Abstract

We describe an LR parser of parts-of-speech (and punctuation labels) for Tree Adjoining Grammars (TAGs), that solves table conflicts in a greedy way, with limited amount of backtracking. We evaluate the parser using the Penn Treebank showing that the method yield very fast parsers with at least reasonable accuracy, confirming the intuition that LR parsing benefits from the use of rich grammars.

1 Introduction

The LR approach for parsing has long been considered for natural language parsing (Lang, 1974; Tomita, 1985; Wright and Wrigley, 1991; Shieber, 1983; Pereira, 1985; Merlo, 1996), but it was not until a more recent past, with the advent of corpus-based techniques made possible by the availability of large treebanks, that parsing results and evaluation started being reported (Briscoe and Carroll, 1993; Inui et al., 1997; Carroll and Briscoe, 1996; Ruland, 2000).

The appeal of LR parsing (Knuth, 1965) derives from its high capacity of postponement of structural decisions, therefore allowing for much of the spurious local ambiguity to be automatically discarded. But it is still the case that conflicts arise in the LR table for natural language grammars, and in large quantity. The key question is how one can use the contextual information contained in the parsing stack to cope with the remaining (local) ambiguity manifested as conflicts in the LR tables. The aforementioned work has concentrated on LR parsing for CFGs which has a clear deficiency in making available sufficient context in the LR states. (Shieber and Johnson, 1993) hints at the relevance of rich grammars on this respect. They use Tree Adjoining Grammars (TAGs) (Joshi and Schabes, 1997; Joshi et al., 1975) to defend the possibility of granular incremental computations in LR parsing. Incidentally or not, they make use of disambiguation contexts that are only possible in a state of a conceptual LR parser for a rich grammar formalism such as TAG, but not for a CFG.

Concrete LR-like algorithms for TAGs have only recently been proposed (Prolo, 2000; Nederhof, 1998), though their evaluation was restricted to the quality of the parsing table (see also (Schabes and Vijay-Shanker, 1990; Kinyon, 1997) for earlier attempts).

In this paper, we revisit the LR parsing technique, applied to a rich grammar formalism: TAG. Following (Briscoe and Carroll, 1993), conflict resolution is based on contextual information extracted from the so called *Instantaneous Description* or *Configuration*: a stack, representing the control memory of the LR parser, and a lookahead sequence, here limited to one symbol.[1]

However, while Briscoe and Carroll invested on massive parallel computation of the possible parsing paths, with pruning and posterior ranking, we ex-

[1] Unlike (Wright and Wrigley, 1991)'s approach who tries to transpose PCFG probabilities to LR tables, facing difficulties which, to the best of our knowledge, have not been yet solved to content (cf. also (Ng and Tomita, 1991; Wright et al., 1991; Abney et al., 1999)).

Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Philadelphia, July 2002, pp. 103-110. Association for Computational Linguistics.

periment with a simple greedy depth-first technique with limited amount of backtracking, that resembles to a certain extent the commitment/recovery models from the psycholinguistic research on human language processing, supported by the occurrence of "garden paths".[2]

We use the Penn Treebank WSJ corpus, release 2 (Marcus et al., 1994), to evaluate the approach.

2 The architecture of the parser

Table 1 shows the architecture of our parsing application. We extract a TAG from a piece of the Penn Treebank, the training corpus, and submit it to an LR parser generator. The same training corpus is used again to extract statistical information that is used by the driver as follows. The grammar generation process generates as a subproduct the TAG derivation trees for the annotated sentences compatible with the extracted grammar trees. This derivation tree is then converted into the sequence of LR parsing actions that would have to be used by the parser to generate exactly that analysis. A parser execution simulation is then performed, guided by the obtained sequence of parsing actions, collecting the statistical information defined in Section 3.

In possession of the LR table, grammar and statistical information, the parser is then able to parse fast natural language sentences annotated for parts-of-speech.

2.1 The extracted grammar

Our target grammar is extracted with a customized version of the extractor defined in (Xia, 2001), which we will not describe here. However, a key aspect to mention is that grammar trees are extracted by factoring of recursion. Even constituents annotated flat in the Treebank are first given a more hierarchical, recursive structure. Therefore the trees generated during parsing will be richer than those in the Treebank. We will return to this point later.

Before grammar extraction, Treebank labels are merged to allow for the generation of a more compact grammar and parsing table, and to concentrate statistical information (e.g., NN and NNS; NNP and NNPS; all labels for finite verb forms). The grammar extractor assigns a plausibility judgment to each extracted tree. When a tree is judged implausible, it is discarded from the grammar, and so are the sentences in the training corpus in which the tree is used. This reduced our training corpus by about 15 %.

[2] See, e.g., (Tanenhaus and Trueswell, 1995) for a survey on human sentence comprehension.

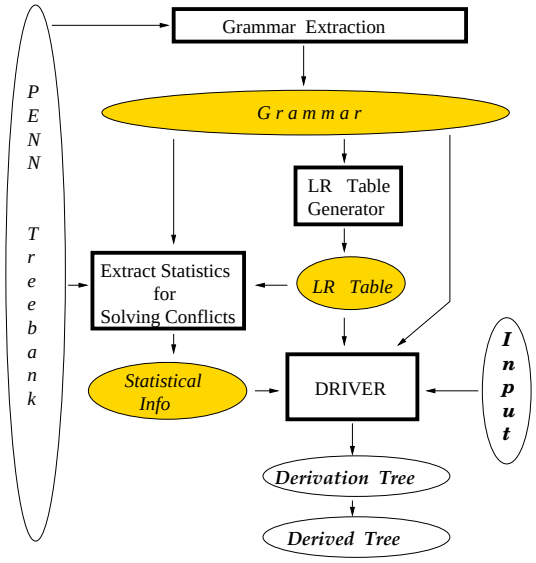


Figure 1: Architecture of the parser

2.2 The LR parser generator

We used the grammar generator in (Prolo, 2000). In this section we only present a parsing example, to illustrate the kinds of action inserted in the generated tables; details concerning how the table is generated are omitted. Consider the TAG fragment in Figure 2 for simple sentences with modal and adverb adjunction. Figure 3 contains a derivation for the sentence *"John will leave soon"*.

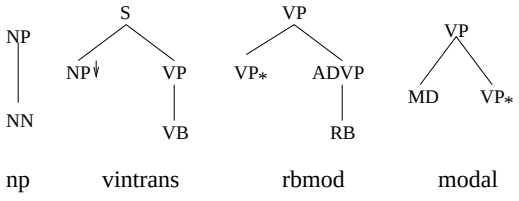
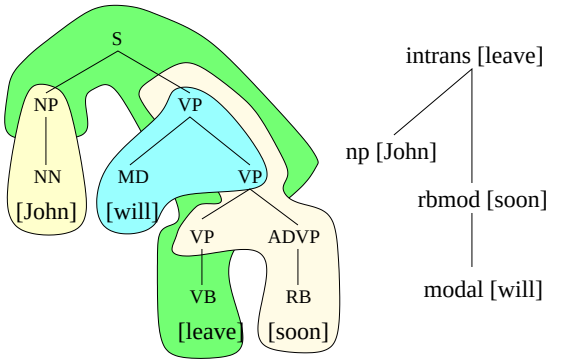


Figure 2: A TAG fragment for simple sentences with VP adjunction

We sketch in Figure 4 the sequence of actions executed by the parser. Technically, each element of the stack would be a pair: the second element being



a) derived tree b) derivation tree

Figure 3: Derivation of *"John will leave soon*

stack	input	sel. action
[]	NN MD ...	shift
[NN]	MD VB ...	reduce **np**
[NP]	MD VB ...	shift
[NP MD]	VB RB $	shift
[NP MD VB]	RB $	bpack VP,1
[NP MD [$_{VP}$ VB]]	RB $	shift
[NP MD [$_{VP}$ VB] RB]	$	bpack VP,2
[NP MD [$_{VP}$ [$_{VP}$ VB] RB]]	$	reduce **modal**
[NP [$_{VP}$ VB] RB]	$	reduce **rbmod**
[NP VB]	$	accept **vintrans**

Figure 4: Parsing *"John/NN will/MD leave/VB soon/RB"* (parsing states were omitted in "stack" for clarity).

an LR state; the first can be either a grammar symbol or an embedded stack. Although the state is the only relevant component of the pair for parsing, in the figure, for presentational purposes, we omit the state and instead show only the symbol/embedded stack component (despite the misleading presence of embedded stacks, actions are executed in constant time). Stacks are surrounded by square brackets. Only the parts of speech have been represented. The ***bpack*** action is not standard in LR parsing. It represents an earlier partial commitment for structure. In its first appearance, it acknowledges that some material will be adjoined to a *VP* that dominates the element at the top of the stack (in fact it dominates the k topmost elements, where k is the second parameter of *bpack*). The material is then enclosed in a substack (the subscript *VP* at the left bracket is for presentation purposes only; that information is in fact in the LR state that would pair with the substack). The next line contains another *bpack* with $k = 2$, that proposes another adjunction, dominating the *VB* and the *RB*. Reductions for auxiliary trees leave no visible trace in the stack after they are executed. The parser executes reductions in a bottom up order with respect to the derivation tree[3]

3 Conflict Resolution

In this session we focus on how to resolve conflicts in the generated parsing table to obtain a single "best" parse for each input sentence. At each step of the parsing process the driver is faced with the task of choosing among a certain number of available actions. At the end, the sequence of actions taken will uniquely define one derivation tree that represents the chosen syntactic analysis.

In our approach, the parser proceeds greedily trying to compute a single successful sequence of actions. Whenever it fails, a backtracking strategy is employed to re-conduce the parser to an alternative path, up to a certain limited number of attempts provided as a parameter to the parser. Choices are made locally. We have not tried to maximize any global measure, such as the probability of the sentence, the probability of a parse given a string, etc.

An *instantaneous description* (or "configuration") can be characterized by two components:

1. The current content of the stack, which includes the current (automaton) state;

2. The suffix of the input not yet shifted into the stack.

The basic parsing approach has two main components:

1. A strategy for ranking the actions available at any given instantaneous description;

2. A greedy strategy for computing the sequence of parsing actions. At each instantaneous description:

   - Choose the highest-ranked action that has not been tried yet and execute it.

[3]Notice that the notion of top-down/bottom-up parsing cannot be defined on the derived tree for TAGs, unless one wants to break the actions into independent subatomic units, e.g., single-level context-free-like expansions.

- If there is no action available, then backtrack: move back to one of the instantaneous descriptions previously entered and choose an alternative action.

3.1 Strategies for ranking conflicting actions

Let $i+1$ be the number of positions in the stack and $q_0, q_1, \ldots, q_{i-2}, q_{i-1}, q_i$ be the sequence of states in the positions.[4] q_i is the current state. Let la be the *lookahead symbol*, the leftmost symbol in the not yet shifted suffix. Let A be the (finite) set of possible actions given by the grammar.

We use two basic ranking functions: $rank_1$ and $rank_2$. The first, naive one considers only the current automaton state (the state at the top of the stack, q_i) and the lookahead symbol, la. It is a poor statistic but it does not suffer of sparse data problems in our training corpus, and hence is used for smoothing. For any *instantaneous description* as described above, we trivially define the $rank_1(a)$, for any action $a \in A$, as a probability estimate for a, given the current state q_i and lookahead symbol la:

$$\begin{aligned} rank_1(a) &= \hat{p}(a|q_i, la) \\ &= count(a, q_i, la)/count(q_i, la), \end{aligned}$$

where $count(n-tuple)$ is the number of times $n-tuple$ occurs in an instantaneous description when parsing the annotated corpus.

It can be observed that individual actions tend to depend on different additional states in the stack. For a *shift* action there is no reason to assume that the previous state is particularly relevant, or that state, say, q_{i-2}, is not. But for a $reduce$ or *bpack* action we should suspect that the state from where the *goto* action is taken is highly relevant. So, for instance, the action $\alpha-$*reduce* t, where the number of non-empty leaves of t is l, would have strong dependency on the state q_{i-l}. An approximation of its rank would be: $rank(a) = \hat{p}(a|q_{i-l}, q_i, la)$. This observation is certainly not new. A similar ranking function is in fact used by Briscoe and Carroll. However, an inconsistency immediately arises: we cannot compare probabilities of events drawn on distinct sample spaces. For instance, if we have two competing actions a_1, an $\alpha-$*reduce* t, and a_2, a *shift*, and we affirm that a_2 depends on q_{i-l}, then, it cannot be true that the *shift* does not depend on q_{i-l}. In fact, it has to be the case that it depends on q_{i-l} as much as a_1. One could suggest calculating the probabilities for all actions conditional to the same set of states, $\{q_{i-l}, q_i\}$. But, in general, we have many more than two actions to decide among. And they are likely to stress their dependencies on different past states. We see that this is not going to work; the number of dependencies, and hence the number of parameters, will grow too big.

A striking solution arises from a notable fact from LR parsing theory for CFGs: If state q_i contains an action *reduce p*, where p is a production with l symbols on its right side, then, the pair (q_{i-l}, q_i), from the instantaneous description, uniquely identifies the entire sequence $q_{i-l}, q_{i-l+1}, ..., q_{i-1}, q_i$. Although this property does not hold for the parser generation algorithm we are using, it is still a good approximation to the true statistical dependencies.[5]

We can use this "approximately correct" property in our benefit: "if state q_i contains an action *reduce* or *bpack* for a number of leaves l, then the dependency on the sequence $q_{i-l}, q_{i-l+1}, ..., q_{i-1}, q_i$ can be approximated by a dependency on the pair (q_{i-l}, q_i)". So a natural candidate for the second state to be considered is the state $q_{i-ml(i)}$, where $ml(i)$ = max$\{l|q_i$ has an action *bpack*(X,l) for some X or q_i has some action *reduce* for a tree with l non-empty leaves $\}$. We define our second ranking function based on that.

$$\begin{aligned} rank_2(a) &= \hat{p}(a|q_i, q_{i-ml(i)}, la) \\ &= \frac{count(a, q_i, q_{i-ml(i)}, la)}{count(q_i, q_{i-ml(i)}, la),} \end{aligned}$$

[4] Recall each position in the stack contains a pair where the second element is an automaton state and the first element is either a symbol or an embedded stack.

[5] That the property does not hold in the algorithm we are using is a consequence of the way the *goto* function for adjunction is defined in (Prolo, 2000), as a function of two states (instead of just a simple transition in the automaton). A detailed argument is beyond the scope of this paper and can be found in (Prolo, 2002), available upon request to the author. That the statement is a good approximation to the true statistical dependencies follows from: (1) adjuncts (that can cause distinct states to intervene between the considered pair), are generally regarded as not restricting the syntactic possibilities of the clause they adjoin to; and (2) in practice, the intermediate states at positions that could be distinct for theoretically different sequences most often have exactly the same characteristics, i.e., they are likely to "accidentally" collapse to the same state.

3.2 The backtracking strategy

We have a (quite narrow) notion of confidence for parsing paths: as long as our sequence of decisions allows the parser to proceed we trust the sequence, and if it has taken us to an acceptance state, we believe we have the correct parse. On the other hand, a crash is our other binary value for confidence, an untrustworthy parsing sequence. In these cases, we know we have made a bad decision somewhere in the path and that we have to start again from that point by following another alternative. This is a backtracking strategy, although not in the common sense of a depth-first walk, (i.e., exploring all the possibilities left before undoing some earlier action). We want to explore strategies of intelligently guessing the restart point.

We use a simple strategy of returning to the decision point that left the highest amount of probability mass unexplored. In order to implement it, we maintain a tree with the entire parsing attempts' history. There will be one path from the root corresponding to the current parsing attempt, the leaf being the current instantaneous description. All other leaves correspond to instantaneous descriptions that have been abandoned (crashing points). If the current leaf crashes, all nodes in the tree (except for the leaves) compete to be the restart point. Keeping all nodes in the tree alive is a direct consequence of the fact that we do not intend to do exhaustive backtracking. We trade space (a tree instead of just a sequence) for time: presumably, by doing smart backtracking we can find a successful path by trying only a fraction of the possible ones. Moreover, we want to find the best (or approximately best) successful path, and a crashing point is a good point to re-evaluate the process. Limits may be added through parameters, so that the parser may give up after a certain amount of attempts or time.

In addition to the instantaneous description, each node contains a record of the alternatives previously tried (the edges to its child nodes in the tree) with their corresponding probabilities, plus a ranked list of the alternatives not yet tried. In particular we maintain the probability mass left unexplored in a node: the sum of the probabilities of the actions not yet tried. Notice that alternatives already tried are indirectly kept alive through their corresponding child nodes.

Let $L(n)$ be the set of actions not yet tried at node n. The probability mass left is $pl(n) = \sum_{a \in L(n)} rank(a, n)$.[6] The backtracking process chooses $a \in L(n)$ for which $rank(a, n)$ is maximum (efficiently maintained using a priority queue). Then we update $pl(n) = pl(n) - rank(a, n)$ and start another branch in the tree by executing a.

4 Evaluation

We evaluated the approach using the Penn Treebank WSJ Corpus, release 2 (Marcus et al., 1994), using Sections 2 to 21 for grammar extraction and training, section 0 for development and 23 for testing.

Only parts-of-speech were used as input.[7] [8]

A smoothed ranking function is defined as follows:

$$rank_{smoo} = \begin{array}{l} \text{if } count(q_i, q_{i-ml(i)}, la) > K \\ \text{then } \; rank_1 \\ \text{else } \; rank_2 \end{array}$$

The best K was experimentally determined to be 1. That is: in general, even if there is minimal evidence of the context including the second state, the statistics using this context lead to a better result than using only one state.

For each sentence there is an initial parsing attempt using only $rank_2$ as the ranking function with an a maximum of 500 backtracking occurrences. If it fails, then the sentence is parsed using $rank_{smoo}$, with a maximum of 3,000 backtracking occurrences.

In table 1 we report the following figures for the development set (Section 0) and test set (Section 23):

- **%failed** is the percentage of sentences for which the parser failed (in the two attempts).

[6] Where $rank(a, n)$ can be any of the ranking functions, at the state n, applied to a. Elsewhere in the paper we have omitted the explicit reference to the state.

[7] However, two new categories were defined: one for time nouns, namely those that appear in the Penn Treebank as heads of constituents marked "TMP" (for temporal); another for the word "that". This is similar to (Collins, 1997)'s and Charniak97's definition of a separate category for auxiliary verbs.

[8] We also included some punctuation symbols among the terminals such as comma, colon and semicolon. They are extracted into the grammar as if they were regular modifiers. Their main use is in guiding parsing decisions.

Section	%failed	tput	recall	prec.	$F_{\beta=1}$
0	1.3	18	81.76		
23	1.9	19	81.41		
0 (flat)	1.3	18	78.21	77.35	77.78
23 (flat)	1.9	19	77.52	76.96	77.24

Table 1: Results on the development and test set

- **recall** and **prec.** are the labeled parsing recall and precision, respectively, as defined in (Collins, 1997) (slightly different from (Black et al., 1991)). $F_{\beta=1}$ is their harmonic average.
- **tput** is the average number of sentences parsed per second. To obtain the average, the number of sentences submitted as input (not only those that parsed successfully) is divided by the total time (excluded the time overhead before it starts parsing the first sentence). The programs were run under Linux, in a PC with a Pentium III 930MHz processor.

The first two lines report the measures for the parsed sentences as originally generated by the parser. We purposefully do not report precision. As we mentioned in the beginning of the paper, the parser assigns to the sentences a much richer hierarchical structure than the Penn Treebank does, which is penalized by the precision measure. The reason for such increase in structure is not quite a particular decision of ours, but a consequence of using a sound grammar under the TAG grammatical formalism.[9]

However, having concluded our *manifesto*, we understand that algorithms that try to keep precision as high as the recall necessarily have losses in recall compared to if they ignored the precision, and therefore in order to have fair comparison with them and to improve the credibility of our results, we flattened the parse trees in a post-processing step, using a simple rule-based technique on top of some frequency measures for individual grammar trees gathered by (Xia, 2001) and the result is presented in the bottom lines of the table.

[9]By sound we mean a grammar that properly factors recursion in one way or another. Grammars have been extracted where the right side of a rule reflects exactly each single-level expansion found in the Penn Treebank. We are also aware of a few alternatives in grammatical formalisms that could capture such flatness, e.g., sister adjunction (Chiang, 2000).

The most salient positive result is that the parser is able to parse sentences at a rate of about 20 sentences per second. Most of the medium-to-high accuracy parsers take at least a few seconds per sentence under the same conditions.[10] This is an enormous speed-up. As for the accuracy, it is not far from the top performing parser for parts-of-speech that we are aware of, reported by (Sima'an, 2000): recall/precision = $80.03/80.99$

Perhaps the most similar work to ours is Briscoe and Carroll's (1993; 1995; 1992; 1996). They implemented a standard LR parser for CFGs, and a probabilistic method for conflict resolution similar to ours in that the decisions are conditioned to the LR states but with different methods. In particular, they proceed in a parallel way accumulating probabilities along the paths and using a Viterbi decoder at the end. Their best published result is of unlabeled bracket recall and precision of 74 % and 73 %, parsing the *Susanne corpus*. Since the unlabeled bracket measures are much easier than the ones we are reporting, on labeled brackets, our results are clearly superior to theirs. Also the Susanne corpus is easier than the Penn Treebank.

There are two additional points we want to make. One is with respect to the ranking function $rank_2$, based on two states. It is a very rich statistic, but suffers from sparse data problems. Parsing section 0 with only this statistics (no form of smoothing), with backtracking limit of 3,000 attempts, we could parse only 31 % of the sentences but the non-flattened recall was 88.33 %, which is quite high for using only parts-of-speech. The second observation is that when parsing with the smoothed function $rank_{smoo}$ most of the sentences use very few number of backtracking attempts. In fact a graph relating number of backtracking attempts k with number of sentences that parse using k attempts shows an $1/x$ relation characteristic of Zipf's law. Most of the time spent with computation is spent with sentences that either fail parsing or parse with difficulty, showing low bracketing accuracy.

[10]The fastest parser we are aware of is from BBN, with a throughput of 3 sentences per second under similar conditions. We also emphasize we have not taken particular care with optimization for speed yet.

5 Conclusions

The results presented here suggest that: (1) the use of a rich grammar as the underlying formalism for the LR techniques makes available enough information to the driver so as to allow for a greedy strategy to achieve reasonable parsing accuracy. (2) LR parsing allows for very fast parsing with at least reasonable accuracy.

The approach seems to have much yet to be explored, mostly to improve the accuracy side. In particular we have not yet come with a solid approach to lexicalization. Using words (as opposed to pos tags) as the terminals of the grammar to be precompiled leads to an explosion in the size of the table: not only the average number of transitions per state grows, but also the number of states itself grows wildly. One very promising approach for a partial solution is to expand the set of terminals by adding some selected syntactic sub-categories that have distinguished syntactic behavior, as we reported in this paper for time nouns, or by individuating frequent words with peculiar behavior, as we did for the word *"that"*. Although we have also done some initial work on a more general approach to clustering words according to their syntactic distribution, they are not still adequate for our purposes. Finally, an earlier simple experiment of adding a dependency on the lookahead's word (recall that la in $rank_1$ and $rank_2$ was the pos tag only), gave us a small improvement of about a couple of percents in the accuracy measures.

A limited amount of parallelism is an important topic to be considered, perhaps together with a better notion (non-binary) of confidence. The high reliability of $rank_2$, suggests that we should look for a way to enrich the parsing table.

LR parser for the full class of TAGs is problematic. The ***bpack*** action of early structural commitment is involved in most of the decision points where the wrong action is taken. We are currently working on a version of the LR parser for a subclass of TAGs, the Tree Insertion Grammars (Schabes and Waters, 1995), for which efficient true LR parsers can be obtained.

References

Steven Abney, David McAllester, and Fernando Pereira. 1999. Relating probabilistic grammar and automata. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, College Park, MD, USA.

Ezra Black, Steven Abney, C. Gdaniec, Ralph Grishman, P. Harrison, Don Hindle, R. Ingria, Fred Jelinek, Judith Klavans, Mark Liberman, Mitchell Marcus, Salim Roukos, Beatrice Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of the DARPA Speech and Natural Language Workshop*, San Mateo, CA, USA.

Ted Briscoe and John Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

Ted Briscoe and John Carroll. 1995. Developing and evaluating a probabilistic LR parser of part-of-speech and punctuation labels. In *Proceedings of the 4th International Workshop on Parsing Technologies (IWPT-95)*, pages 48–58, Prague/Karlovy Vary, Czech Republic.

John Carroll and Ted Briscoe. 1992. Probabilistic normalisation and unpacking of packed parse forests for unification-based grammars. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, Cambridge, MA, USA.

John Carroll and Ted Briscoe. 1996. Apportioning development effort in a probabilistic LR parsing system through evaluation. In *Proceedings of the Conference on Empirical Methods in NLP*, pages 92–100, Philadelphia, PA, USA.

David Chiang. 2000. Statistical parsing with an automatically-extracted Tree Adjoining Grammar. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, China.

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

Kentaro Inui, Virach Sornlertlamvanich, Hozumi Tanaka, and Takenobu Tokunaga. 1997. A new formalization of probabilistic GLR parsing. In *Proceedings of the 5th International Workshop on Parsing Technologies (IWPT-97)*, Cambridge, MA, USA.

Aravind K. Joshi and Yves Schabes. 1997. Tree-Adjoining Grammars. In *Handbook of Formal Languages*, volume 3, pages 69–123. Springer-Verlag, Berlin.

Aravind K. Joshi, L. Levy, and M. Takahashi. 1975. Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 10(1).

Alexandra Kinyon. 1997. Un algorithme d'analyse LR(0) pour les grammaires d'arbres adjoints lexicaliseées. In D. Genthial, editor, *Quatrième conférence annuelle sur Le Traitement Automatique du Langage Naturel, Actes*, pages 93–102, Grenoble, France.

Donald E. Knuth. 1965. On the translation of languages from left to right. *Information and Control*, 8(6):607–639.

Bernard Lang. 1974. Deterministic techniques for efficient non-deterministic parsers. In *Automata, Languages and Programming, 2nd Colloquium*, volume 14 of *Lecture Notes in Computer Science*, pages 255–269, Saarbrücken. Springer-Verlag, Berlin.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of the 1994 Human Language Technology Workshop*.

Paola Merlo. 1996. *Parsing with Principles and Classes of Information*. Kluwer Academic Publishers, Boston, MA, USA.

Mark-Jan Nederhof. 1998. An alternative LR algorithm for TAGs. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 16th International Conference on Computational Linguistics*, Montreal, Canada.

See-Kiong Ng and Masaru Tomita. 1991. Probabilistic LR parsing for Generalized context-free grammars. In *Proceedings of the Second International Workshop on Parsing Technologies (IWPT-91)*, Cancun, Mexico.

Fernando Pereira. 1985. A new characterization of attachment preferences. In David R. Dowty, Lauri Kartunen, and Arnold M. Zwicky, editors, *Natural Language Parsing: Psychological, computational, and theoretical perspectives*, pages 307–319. Cambridge University Press, New York, NY, USA.

Carlos A. Prolo. 2000. An efficient LR parser generator for Tree Adjoining Grammars. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT-2000)*, Trento, Italy.

Carlos A. Prolo. 2002. LR parsing for Tree Adjoioning Grammars and its application to corpus-based natural language parsing. Ph.D. thesis proposal, University of Pennsylvania.

Tobias Ruland. 2000. A context-sensitive model for probabilistic LR parsing of spoken language with transformation-based postprocessing. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING'2000)*, pages 677–683, Saarbrücken, Germany.

Y. Schabes and K. Vijay-Shanker. 1990. Deterministic left to right parsing of tree adjoining languages. In *Proceedings of 28th Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Pittsburgh, Pennsylvania, USA.

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(4):479–513.

Stuart Shieber and Mark Johnson. 1993. Variations on incremental interpretation. *Journal of Psycholinguistic Research*, 22(2):287–318.

Stuart M. Shieber. 1983. Sentence disambiguation by a Shift-Reduce parsing technique. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 119–122, Cambridge, MA, USA.

Khalil Sima'an. 2000. Tree-gram parsing: Lexical dependencies and structural relations. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, China.

Michael K. Tanenhaus and John C. Trueswell. 1995. Sentence comprehension. In Joanne L. Miller and Peter D. Eiwas, editors, *Speech, Language, and Communication*, pages 217–262. Academic Press, San Diego, CA, USA.

Masaru Tomita. 1985. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, Boston, MA, USA.

J. H. Wright and E. N. Wrigley. 1991. GLR parsing with probability. In Masaru Tomita, editor, *Generalized LR Parsing*, pages 113–128. Kluwer Academic Publishers, Boston, MA, USA.

Jerry Wright, Ave Wrigley, and Richard Sharman. 1991. Adaptive probabilistic Generalized LR parsing. In *Proceedings of the Second International Workshop on Parsing Technologies (IWPT-91)*, Cancun, Mexico.

Fei Xia. 2001. *Investigating the Relationship between Grammars and Treebanks for Natural Languages*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.