Efficient probabilistic top-down and left-corner parsing†

Brian Roark and Mark Johnson
Cognitive and Linguistic Sciences
Box 1978, Brown University
Providence, RI 02912, USA
brian-roark@brown.edu mj@cs.brown.edu

Abstract

This paper examines efficient predictive broad-coverage parsing without dynamic programming. In contrast to bottom-up methods, depth-first top-down parsing produces partial parses that are fully connected trees spanning the entire left context, from which any kind of non-local dependency or partial semantic interpretation can in principle be read. We contrast two predictive parsing approaches, top-down and left-corner parsing, and find both to be viable. In addition, we find that enhancement with non-local information not only improves parser accuracy, but also substantially improves the search efficiency.

1 Introduction

Strong empirical evidence has been presented over the past 15 years indicating that the human sentence processing mechanism makes *on-line* use of contextual information in the preceding discourse (Crain and Steedman, 1985; Altmann and Steedman, 1988; Britt, 1994) and in the visual environment (Tanenhaus et al., 1995). These results lend support to Mark Steedman's (1989) "intuition" that sentence interpretation takes place incrementally, and that partial interpretations are being built while the sentence is being perceived. This is a very commonly held view among psycholinguists today.

Many possible models of human sentence processing can be made consistent with the above view, but the general assumption that must underlie them all is that explicit relationships between lexical items in the sentence must be specified incrementally. Such a processing mechanism stands in marked contrast to dynamic programming parsers, which delay construction of a constituent until all of its sub-constituents have been completed, and whose partial parses thus consist of disconnected tree fragments. For example, such parsers do not integrate a main verb into the same tree structure as its subject NP until the VP has been completely parsed, and in many cases this is the final step of the entire parsing process. Without explicit on-line integration, it would be difficult (though not impossible) to produce partial interpretations on-line. Similarly, it may be difficult to use non-local statistical dependencies (e.g. between subject and main verb) to actively guide such parsers.

Our predictive parser does not use dynamic programming, but rather maintains fully connected trees spanning the entire left context, which make explicit the relationships between constituents required for partial interpretation. The parser uses probabilistic best-first parsing methods to pursue the most likely analyses first, and a beam-search to avoid the non-termination problems typical of non-statistical top-down predictive parsers.

There are two main results. First, this approach works and, with appropriate attention to specific algorithmic details, is surprisingly efficient. Second, not just accuracy but also efficiency improves as the language model is made more accurate. This bodes well for future research into the use of other non-local (e.g. lexical and semantic) information to guide the parser.

In addition, we show that the improvement in accuracy associated with left-corner parsing over top-down is attributable to the non-local information supplied by the strategy, and can thus be obtained through other methods that utilize that same information.

†This material is based on work supported by the National Science Foundation under Grant No. SBR-9720368.

2 Parser architecture

The parser proceeds incrementally from left to right, with one item of look-ahead. Nodes are expanded in a standard top-down, left-to-right fashion. The parser utilizes: (i) a probabilistic context-free grammar (PCFG), induced via standard relative frequency estimation from a corpus of parse trees; and (ii) look-ahead probabilities as described below. Multiple competing partial parses (or analyses) are held on a priority queue, which we will call the *pending* heap. They are ranked by a figure of merit (FOM), which will be discussed below. Each analysis has its own stack of nodes to be expanded, as well as a history, probability, and FOM. The highest ranked analysis is popped from the pending heap, and the category at the top of its stack is expanded. A category is expanded using every rule which could eventually reach the look-ahead terminal. For every such rule expansion, a new analysis is created[1] and pushed back onto the pending heap.

The FOM for an analysis is the product of the probabilities of all PCFG rules used in its derivation and what we call its look-ahead probability (LAP). The LAP approximates the product of the probabilities of the rules that will be required to link the analysis in its current state with the look-ahead terminal[2]. That is, for a grammar G, a stack state $[C_1 \ldots C_n]$ and a look-ahead terminal item ω:

$$(1)\ LAP = P_G([C_1 \ldots C_n] \xrightarrow{\star} \omega\alpha)$$

We recursively estimate this with two empirically observed conditional probabilities for every non-terminal C_i on the stack: $\widehat{P}(C_i \xrightarrow{\star} \omega)$ and $\widehat{P}(C_i \xrightarrow{\star} \epsilon)$. The LAP approximation for a given stack state and look-ahead terminal is:

$$(2)\ P_G([C_i \ldots C_n] \xrightarrow{\star} \omega\alpha) \approx \widehat{P}(C_i \xrightarrow{\star} \omega) + \widehat{P}(C_i \xrightarrow{\star} \epsilon) * P_G([C_{i+1} \ldots C_n] \xrightarrow{\star} \omega\alpha)$$

When the topmost stack category of an analysis matches the look-ahead terminal, the terminal is popped from the stack and the analysis is pushed onto a second priority queue, which we will call the *success* heap. Once there are "enough" analyses on the success heap, all those remaining on the pending heap are discarded. The success heap then becomes the pending heap, and the look-ahead is moved forward to the next item in the input string. When the end of the input string is reached, the analysis with the highest probability and an empty stack is returned as the parse. If no such parse is found, an error is returned.

The specifics of the beam-search dictate how many analyses on the success heap constitute "enough". One approach is to set a constant beam width, e.g. 10,000 analyses on the success heap, at which point the parser moves to the next item in the input. A problem with this approach is that parses towards the bottom of the success heap may be so unlikely relative to those at the top that they have little or no chance of becoming the most likely parse at the end of the day, causing wasted effort. An alternative approach is to dynamically vary the beam width by stipulating a factor, say 10^{-5}, and proceed until the best analysis on the pending heap has an FOM less than 10^{-5} times the probability of the best analysis on the success heap. Sometimes, however, the number of analyses that fall within such a range can be enormous, creating nearly as large of a processing burden as the first approach. As a compromise between these two approaches, we stipulated a base beam factor α (usually 10^{-4}), and the actual beam factor used was $\alpha * \beta$, where β is the number of analyses on the success heap. Thus, when β is small, the beam stays relatively wide, to include as many analyses as possible; but as β grows, the beam narrows. We found this to be a simple and successful compromise.

Of course, with a left recursive grammar, such a top-down parser may never terminate. If *no* analysis ever makes it to the success heap, then, however one defines the beam-search, a top-down depth-first search with a left-recursive grammar will never terminate. To avoid this, one must place an upper bound on the number of analyses allowed to be pushed onto the pending heap. If that bound is exceeded, the parse fails. With a left-corner strategy, which is not prey to left recursion, no such upper bound is necessary.

[1] We count each of these as a parser state (or rule expansion) *considered*, which can be used as a measure of efficiency.

[2] Since this is a non-lexicalized grammar, we are taking pre-terminal POS markers as our terminal items.

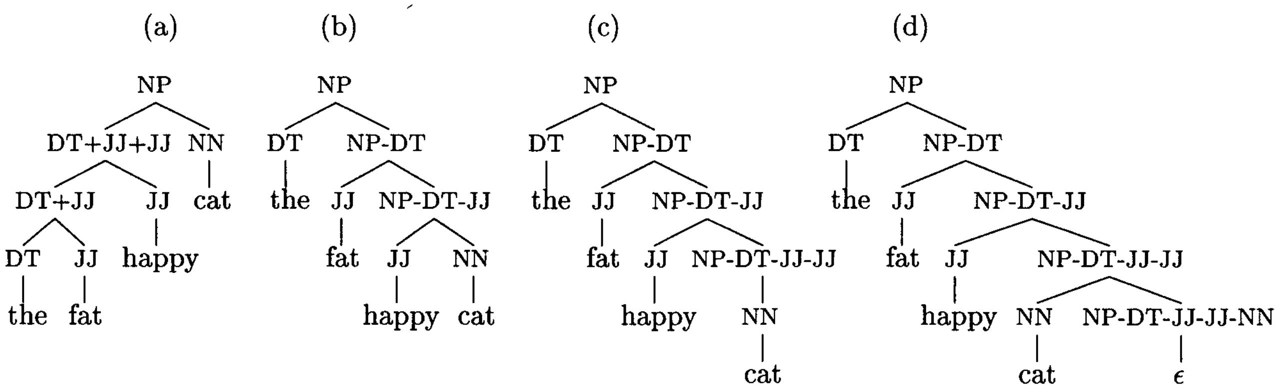

Figure 1: Binarized trees: (a) left binarized (LB); (b) right binarized to binary (RB2); (c) right binarized to unary (RB1); (d) right binarized to nullary (RB0)

3 Grammar transforms

Nijholt (1980) characterized parsing strategies in terms of *announce points*: the point at which a parent category is announced (identified) relative to its children, and the point at which the rule expanding the parent is identified. In pure top-down parsing, a parent category and the rule expanding it are announced *before* any of its children. In pure bottom-up parsing, they are identified *after* all of the children. Grammar transforms are one method for changing the announce points. In top-down parsing with an appropriately binarized grammar, the parent is identified *before*, but the rule expanding the parent *after*, all of the children. Left-corner parsers announce a parent category and its expanding rule *after* its leftmost child has been completed, but *before* any of the other children.

3.1 Delaying rule identification through binarization

Suppose that the category on the top of the stack is an NP and there is a determiner (DT) in the look-ahead. In such a situation, there is no information to distinguish between the rules $NP \rightarrow DT \;\; JJ \;\; NN$ and $NP \rightarrow DT \;\; JJ \;\; NNS$. If the decision can be delayed, however, until such a time as the relevant pre-terminal is in the look-ahead, the parser can make a more informed decision. Grammar binarization is one way to do this, by allowing the parser to use a rule like $NP \rightarrow DT \;\; NP\text{-}DT$, where the new non-terminal $NP\text{-}DT$ can expand into anything that follows a DT in an NP. The expansion of $NP\text{-}DT$ occurs only after the next pre-terminal is in the look-ahead. Such a delay is essential for an efficient implementation of the kind of incremental parser that we are proposing.

There are actually several ways to make a grammar binary, some of which are better than others for our parser. The first distinction that can be drawn is between what we will call *left* binarization (LB) versus *right* binarization (RB, see figure 1). In the former, the leftmost items on the righthand-side of each rule are grouped together; in the latter, the rightmost items on the righthand-side of the rule are grouped together. Notice that, for a top-down, left-to-right parser, RB is the appropriate transform, because it underspecifies the right siblings. With LB, a top-down parser must identify all of the siblings before reaching the leftmost item, which does not aid our purposes.

Within RB transforms, however, there is some variation, with respect to how long rule underspecification is maintained. One method is to have the final underspecified category rewrite as a binary rule (hereafter RB2, see figure 1b). Another is to have the final underspecified category rewrite as a unary rule (RB1, figure 1c). The last is to have the final underspecified category rewrite as a nullary rule (RB0, figure 1d). Notice that the original motivation for RB, to delay specification until the relevant items are present in the look-ahead, is not served by RB2, because the second child must be specified without being present in the look-ahead. RB0 pushes the look-ahead out to the first item in the string *after* the constituent being expanded, which can be useful in deciding between rules of unequal length, e.g. $NP \rightarrow DT \;\; NN$ and $NP \rightarrow DT \;\; NN \;\; NN$.

Table 1 summarizes some trials demonstrat-

Binarization	Rules in Grammar	Percent of Sentences Parsed*	Avg. States Considered	Avg. Labelled Precision and Recall†	Avg. MLP Labelled Prec/Rec†	Ratio of Avg. Prob to Avg. MLP Prob†
None	14962	34.16	19270	.65521	.76427	.001721
LB	37955	33.99	96813	.65539	.76095	.001440
RB1	29851	91.27	10140	.71616	.72712	.340858
RB0	41084	97.37	13868	.73207	.72327	.443705

Beam Factor = 10^{-4} *Length $\leq$ 40 (2245 sentences in F23 - Avg. length = 21.68) †Of those sentences parsed

Table 1: The effect of different approaches to binarization

ing the effect of different binarization approaches on parser performance. The grammars were induced from sections 2-21 of the Penn Wall St. Journal Treebank (Marcus et al., 1993), and tested on section 23. For each transform tested, every tree in the training corpus was transformed before grammar induction, resulting in a transformed PCFG and look-ahead probabilities estimated in the standard way. Each parse returned by the parser was de-transformed for evaluation[3]. The parser used in each trial was identical, with a base beam factor $\alpha = 10^{-4}$. The performance is evaluated using these measures: (i) the percentage of candidate sentences for which a parse was found (coverage); (ii) the average number of states (i.e. rule expansions) considered per candidate sentence (efficiency); and (iii) the average labelled precision and recall of those sentences for which a parse was found (accuracy). We also used the same grammars with an exhaustive, bottom-up CKY parser, to ascertain both the accuracy and probability of the maximum likelihood parse (MLP). We can then additionally compare the parser's performance to the MLP's on those same sentences.

As expected, *left* binarization conferred no benefit to our parser. *Right* binarization, in contrast, improved performance across the board. RB0 provided a substantial improvement in coverage and accuracy over RB1, with something of a decrease in efficiency. This efficiency hit is partly attributable to the fact that the same tree has more nodes with RB0. Indeed, the efficiency improvement with right binarization over the standard grammar is even more interesting in light of the great increase in the size of the grammars.

It is worth noting at this point that, with the RB0 grammar, this parser is now a viable broad-coverage statistical parser, with good coverage, accuracy, and efficiency[4]. Next we considered the left-corner parsing strategy.

3.2 Left-corner parsing

Left-corner (LC) parsing (Rosenkrantz and Lewis II, 1970) is a well-known strategy that uses both bottom-up evidence (from the left corner of a rule) and top-down prediction (of the rest of the rule). Rosenkrantz and Lewis showed how to transform a context-free grammar into a grammar that, when used by a top-down parser, follows the same search path as an LC parser. These LC grammars allow us to use exactly the same predictive parser to evaluate top-down versus LC parsing. Naturally, an LC grammar performs best with our parser when right binarized, for the same reasons outlined above. We use transform composition to apply first one transform, then another to the output of the first. We denote this A $\circ$ B where (A $\circ$ B)(t) = B (A (t)). After applying the left-corner transform, we then binarize the resulting grammar[5], i.e. LC $\circ$ RB.

Another probabilistic LC parser investigated (Manning and Carpenter, 1997), which utilized an LC parsing architecture (not a transformed grammar), also got a performance boost

[3]See Johnson (1998) for details of the transform/de-transform paradigm.

[4]The very efficient bottom-up statistical parser detailed in Charniak et al. (1998) measured efficiency in terms of total edges *popped*. An edge (or, in our case, a parser state) is *considered* when a probability is calculated for it, and we felt that this was a better efficiency measure than simply those popped. As a baseline, their parser *considered* an average of 2216 edges per sentence in section 22 of the WSJ corpus (p.c.).

[5]Given that the LC transform involves nullary productions, the use of RB0 is not needed, i.e. nullary productions need only be introduced from one source. Thus binarization with left corner is always to unary (RB1).

Transform	Rules in Grammar	Pct. of Sentences Parsed*	Avg. States Considered	Avg Labelled Precision and Recall†	Avg. MLP Labelled Prec/Rec†	Ratio of Avg. Prob to Avg. MLP Prob†
Left Corner (LC)	21797	91.75	9000	.76399	.78156	.175928
LB ∘ LC	53026	96.75	7865	.77815	.78056	.359828
LC ∘ RB	53494	96.7	8125	.77830	.78066	.359439
LC ∘ RB ∘ ANN	55094	96.21	7945	.77854	.78094	.346778
RB ∘ LC	86007	93.38	4675	.76120	.80529	.267330

Beam Factor = 10^{-4} *Length $\leq$ 40 (2245 sentences in F23 - Avg. length = 21.68) †Of those sentences parsed

Table 2: Left Corner Results

through right binarization. This, however, is equivalent to RB ∘ LC, which is a very different grammar from LC ∘ RB. Given our two binarization orientations (LB and RB), there are four possible compositions of binarization and LC transforms:

(a) LB ∘ LC (b) RB ∘ LC (c) LC ∘ LB (d) LC ∘ RB

Table 2 shows left-corner results over various conditions[6]. Interestingly, options (a) and (d) encode the same information, leading to nearly identical performance[7]. As stated before, right binarization moves the rule announce point from before to after all of the children. The LC transform is such that LC ∘ RB also delays *parent* identification until after all of the children. The transform LC ∘ RB ∘ ANN moves the parent announce point back to the left corner by introducing unary rules at the left corner that simply identify the parent of the binarized rule. This allows us to test the effect of the position of the parent announce point on the performance of the parser. As we can see, however, the effect is slight, with similar performance on all measures.

RB ∘ LC performs with higher accuracy than the others when used with an exhaustive parser, but seems to require a massive beam in order to even approach performance at the MLP level. Manning and Carpenter (1997) used a beam width of 40,000 parses on the success heap at each input item, which must have resulted in an order of magnitude more rule expansions than what we have been considering up to now, and yet their average labelled precision and recall (.7875) still fell well below what we found to be the MLP accuracy (.7987) for the grammar. We are still investigating why this grammar functions so poorly when used by an incremental parser.

3.3 Non-local annotation

Johnson (1998) discusses the improvement of PCFG models via the annotation of non-local information onto non-terminal nodes in the trees of the training corpus. One simple example is to copy the parent node onto every non-terminal, e.g. the rule $S \rightarrow NP\ VP$ becomes $S \rightarrow NP^{\uparrow}S\ VP^{\uparrow}S$. The idea here is that the distribution of rules of expansion of a particular non-terminal may differ depending on the non-terminal's parent. Indeed, it was shown that this additional information improves the MLP accuracy dramatically.

We looked at two kinds of non-local information annotation: parent (PA) and left-corner (LCA). Left-corner parsing gives improved accuracy over top-down or bottom-up parsing with the same grammar. Why? One reason may be that the ancestor category exerts the same kind of non-local influence upon the parser that the parent category does in parent annotation. To test this, we annotated the left-corner ancestor category onto every leftmost non-terminal category. The results of our annotation trials are shown in table 3.

There are two important points to notice from these results. First, with PA we get not only the previously reported improvement in accuracy, but additionally a fairly dramatic decrease in the number of parser states that must be visited to find a parse. That is, the non-local information not only improves the final product of the parse, but it guides the parser more quickly

[6] Option (c) is not the appropriate kind of binarization for our parser, as argued in the previous section, and so is omitted.

[7] The difference is due to the introduction of vacuous unary rules with RB.

Transform	Rules in Grammar	Pct. of Sentences Parsed*	Avg. States Considered	Avg Labelled Precision and Recall†	Avg. MLP Labelled Prec/Rec†	Ratio of Avg. Prob to Avg. MLP Prob†
RB0	41084	97.37	13868	.73207	.72327	.443705
PA ∘ RB0	63467	95.19	8596	.79188	.79759	.486995
LC ∘ RB	53494	96.7	8125	.77830	.78066	.359439
LCA ∘ RB0	58669	96.48	11158	.77476	.78058	.495912
PA ∘ LC ∘ RB	80245	93.52	4455	.81144	.81833	.484428

Beam Factor = 10^{-4} *Length $\leq$ 40 (2245 sentences in F23 - Avg. length = 21.68) †Of those sentences parsed

Table 3: Non-local annotation results

to the final product. The annotated grammar has 1.5 times as many rules, and would slow a bottom-up CKY parser proportionally. Yet our parser actually considers far fewer states en route to the more accurate parse.

Second, LC-annotation gives nearly all of the accuracy gain of left-corner parsing[8], in support of the hypothesis that the ancestor information was responsible for the observed accuracy improvement. This result suggests that if we can determine the information that is being annotated by the troublesome RB ∘ LC transform, we may be able to get the accuracy improvement with a relatively narrow beam. Parent-annotation before the LC transform gave us the best performance of all, with very few states considered on average, and excellent accuracy for a non-lexicalized grammar.

4 Accuracy/Efficiency tradeoff

One point that deserves to be made is that there is something of an accuracy/efficiency tradeoff with regards to the base beam factor. The results given so far were at 10^{-4}, which functions pretty well for the transforms we have investigated. Figures 2 and 3 show four performance measures for four of our transforms at base beam factors of 10^{-3}, 10^{-4}, 10^{-5}, and 10^{-6}. There is a dramatically increasing efficiency burden as the beam widens, with varying degrees of payoff. With the top-down transforms (RB0 and PA ∘ RB0), the ratio of the average probability to the MLP probability does improve substantially as the beam grows, yet with only marginal improvements in coverage and accuracy. Increasing the beam seems to do less with the left-corner transforms.

[8]The rest could very well be within noise.

5 Conclusions and Future Research

We have examined several probabilistic predictive parser variations, and have shown the approach in general to be a viable one, both in terms of the quality of the parses, and the efficiency with which they are found. We have shown that the improvement of the grammars with non-local information not only results in better parses, but guides the parser to them much more efficiently, in contrast to dynamic programming methods. Finally, we have shown that the accuracy improvement that has been demonstrated with left-corner approaches can be attributed to the non-local information utilized by the method.

This is relevant to the study of the human sentence processing mechanism insofar as it demonstrates that it is possible to have a model which makes explicit the syntactic relationships between items in the input incrementally, while still scaling up to broad-coverage.

Future research will include:

- lexicalization of the parser
- utilization of fully connected trees for additional syntactic and semantic processing
- the use of syntactic predictions in the beam for language modeling
- an examination of predictive parsing with a left-branching language (e.g. German)

In addition, it may be of interest to the psycholinguistic community if we introduce a time variable into our model, and use it to compare such competing sentence processing models as race-based and competition-based parsing.

References

G. Altmann and M. Steedman. 1988. Interaction with context during human sentence processing. *Cognition*, 30:198–238.

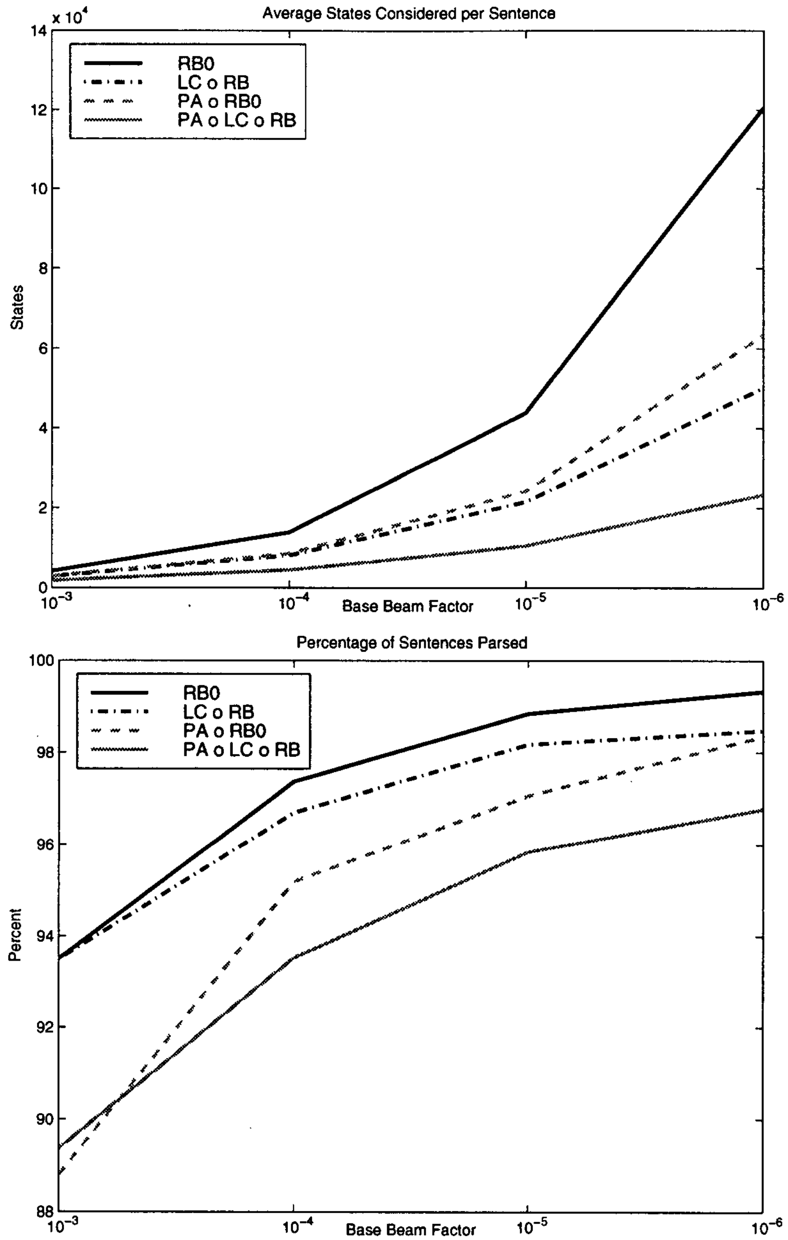

Figure 2: Changes in performance with beam factor variation

M. Britt. 1994. The interaction of referential ambiguity and argument structure. *Journal of Memory and Language*, 33:251–283.

E. Charniak, S. Goldwater, and M. Johnson. 1998. Edge-based best-first chart parsing. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 127–133.

S. Crain and M. Steedman. 1985. On not being led up the garden path: The use of context by the psychological parser. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing*. Cambridge University Press, Cambridge, UK.

M. Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24:617–636.

C. Manning and B. Carpenter. 1997. Probabilistic parsing using left corner language models. In *Proceedings of the Fifth International Workshop on Parsing Technologies.*

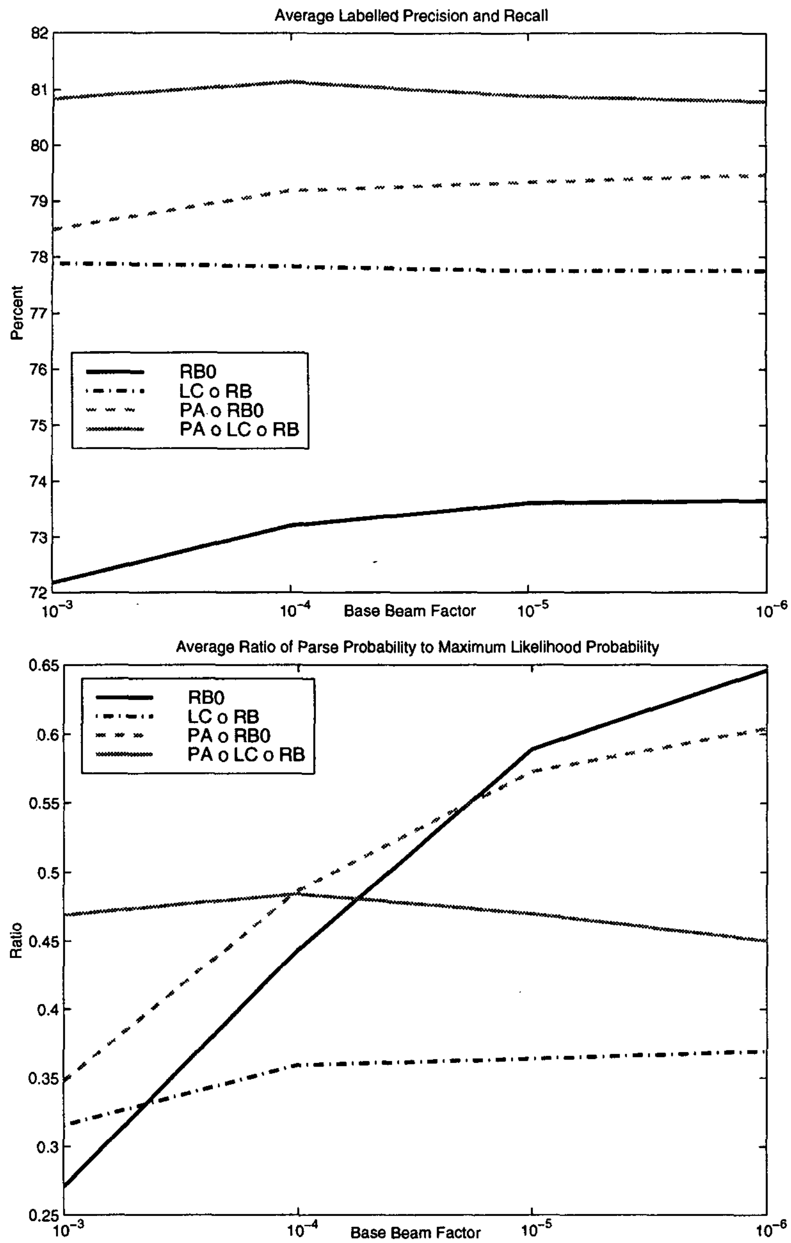

Figure 3: Changes in performance with beam factor variation

M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

A. Nijholt. 1980. *Context-free Grammars: Covers, Normal Forms, and Parsing.* Springer Verlag, Berlin.

S.J. Rosenkrantz and P.M. Lewis II. 1970. Deterministic left corner parsing. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata*, pages 139–152.

M. Steedman. 1989. Grammar, interpretation, and processing from the lexicon. In W. Marslen-Wilson, editor, *Lexical representation and process.* MIT Press, Cambridge, MA.

M. Tanenhaus, M. Spivey-Knowlton, K. Eberhard, and J. Sedivy. 1995. Integration of visual and linguistic information during spoken language comprehension. *Science*, 268:1632–1634.