To appear in *Proceedings of the 34th Annual Meeting of the ACL*, June 1996

Parsing Algorithms and Metrics

Joshua Goodman
Harvard University
33 Oxford St.
Cambridge, MA 02138
goodman@das.harvard.edu

Abstract

Many different metrics exist for evaluating parsing results, including Viterbi, Crossing Brackets Rate, Zero Crossing Brackets Rate, and several others. However, most parsing algorithms, including the Viterbi algorithm, attempt to optimize the same metric, namely the probability of getting the correct labelled tree. By choosing a parsing algorithm appropriate for the evaluation metric, better performance can be achieved. We present two new algorithms: the "Labelled Recall Algorithm," which maximizes the expected Labelled Recall Rate, and the "Bracketed Recall Algorithm," which maximizes the Bracketed Recall Rate. Experimental results are given, showing that the two new algorithms have improved performance over the Viterbi algorithm on many criteria, especially the ones that they optimize.

1 Introduction

In corpus-based approaches to parsing, one is given a treebank (a collection of text annotated with the "correct" parse tree) and attempts to find algorithms that, given unlabelled text from the treebank, produce as similar a parse as possible to the one in the treebank.

Various methods can be used for finding these parses. Some of the most common involve inducing Probabilistic Context-Free Grammars (PCFGs), and then parsing with an algorithm such as the Labelled Tree (Viterbi) Algorithm, which maximizes the probability that the output of the parser (the "guessed" tree) is the one that the PCFG produced. This implicitly assumes that the induced PCFG does a good job modeling the corpus.

There are many different ways to evaluate these parses. The most common include the Labelled Tree Rate (also called the Viterbi Criterion or Exact Match Rate), Consistent Brackets Recall Rate (also called the Crossing Brackets Rate), Consistent Brackets Tree Rate (also called the Zero Crossing Brackets Rate), and Precision and Recall. Despite the variety of evaluation metrics, nearly all researchers use algorithms that maximize performance on the Labelled Tree Rate, even in domains where they are evaluating using other criteria.

We propose that by creating algorithms that optimize the evaluation criterion, rather than some related criterion, improved performance can be achieved.

In Section 2, we define most of the evaluation metrics used in this paper and discuss previous approaches. Then, in Section 3, we discuss the Labelled Recall Algorithm, a new algorithm that maximizes performance on the Labelled Recall Rate. In Section 4, we discuss another new algorithm, the Bracketed Recall Algorithm, that maximizes performance on the Bracketed Recall Rate (closely related to the Consistent Brackets Recall Rate). Finally, we give experimental results in Section 5 using these two algorithms in appropriate domains, and compare them to the Labelled Tree (Viterbi) Algorithm, showing that each algorithm generally works best when evaluated on the criterion that it optimizes.

2 Evaluation Metrics

In this section, we first define basic terms and symbols. Next, we define the different metrics used in evaluation. Finally, we discuss the relationship of these metrics to parsing algorithms.

2.1 Basic Definitions

Let w_a denote word a of the sentence under consideration. Let w_a^b denote $w_a w_{a+1} ... w_{b-1} w_b$; in particular let w_1^n denote the entire sequence of terminals (words) in the sentence under consideration.

In this paper we assume all guessed parse trees are binary branching. Let a parse tree T be defined as a set of triples (s, t, X)—where s denotes the position of the first symbol in a constituent, t denotes the position of the last symbol, and X represents a terminal or nonterminal symbol—meeting the following three requirements:

- The sentence was generated by the start symbol, S. Formally, $(1, n, S) \in T$.
- Every word in the sentence is in the parse tree. Formally, for every s between 1 and n the triple $(s, s, w_s) \in T$.
- The tree is binary branching and consistent. Formally, for every (s, t, X) in T, $s \neq t$, there is exactly one r, Y, and Z such that $s \leq r < t$ and $(s, r, Y) \in T$ and $(r+1, t, Z) \in T$.

Let T_C denote the "correct" parse (the one in the treebank) and let T_G denote the "guessed" parse (the one output by the parsing algorithm). Let N_G denote $|T_G|$, the number of nonterminals in the guessed parse tree, and let N_C denote $|T_C|$, the number of nonterminals in the correct parse tree.

2.2 Evaluation Metrics

There are various levels of strictness for determining whether a constituent (element of T_G) is "correct." The strictest of these is Labelled Match. A constituent $(s, t, X) \in T_G$ is correct according to Labelled Match if and only if $(s, t, X) \in T_C$. In other words, a constituent in the guessed parse tree is correct if and only if it occurs in the correct parse tree.

The next level of strictness is Bracketed Match. Bracketed match is like labelled match, except that the nonterminal label is ignored. Formally, a constituent $(s, t, X) \in T_G$ is correct according to Bracketed Match if and only if there exists a Y such that $(s, t, Y) \in T_C$.

The least strict level is Consistent Brackets (also called Crossing Brackets). Consistent Brackets is like Bracketed Match in that the label is ignored. It is even less strict in that the observed (s, t, X) need not be in T_C—it must simply not be ruled out by any $(q, r, Y) \in T_C$. A particular triple (q, r, Y) rules out (s, t, X) if there is no way that (s, t, X) and (q, r, Y) could both be in the same parse tree. In particular, if the interval (s, t) crosses the interval (q, r), then (s, t, X) is ruled out and counted as an error. Formally, we say that (s, t) crosses (q, r) if and only if $s < q \leq t < r$ or $q < s \leq r < t$.

If T_C is binary branching, then Consistent Brackets and Bracketed Match are identical. The following symbols denote the number of constituents that match according to each of these criteria.

$L = |T_C \cap T_G|$: the number of constituents in T_G that are correct according to Labelled Match.
$B = |\{(s, t, X) : (s, t, X) \in T_G$ and for some $Y\ (s, t, Y) \in T_C\}|$: the number of constituents in T_G that are correct according to Bracketed Match.

$C = |\{(s, t, X) \in T_G :$ there is no $(v, w, Y) \in T_C$ crossing $(s, t)\}|$: the number of constituents in T_G correct according to Consistent Brackets.

Following are the definitions of the six metrics used in this paper for evaluating binary branching trees:

(1) *Labelled Recall Rate* $= L/N_C$.

(2) *Labelled Tree Rate* $=$ 1 if $L = N_C$. It is also called the Viterbi Criterion.

(3) *Bracketed Recall Rate* $= B/N_C$.

(4) *Bracketed Tree Rate* $=$ 1 if $B = N_C$.

(5) *Consistent Brackets Recall Rate* $= C/N_G$. It is often called the Crossing Brackets Rate. In the case where the parses are binary branching, this criterion is the same as the Bracketed Recall Rate.

(6) *Consistent Brackets Tree Rate* $=$ 1 if $C = N_G$. This metric is closely related to the Bracketed Tree Rate. In the case where the parses are binary branching, the two metrics are the same. This criterion is also called the Zero Crossing Brackets Rate.

The preceding six metrics each correspond to cells in the following table:

	Recall	Tree
Consistent Brackets	C/N_G	1 if $C = N_G$
Brackets	B/N_C	1 if $B = N_C$
Labels	L/N_C	1 if $L = N_C$

2.3 Maximizing Metrics

Despite this long list of possible metrics, there is only one metric most parsing algorithms attempt to maximize, namely the Labelled Tree Rate. That is, most parsing algorithms assume that the test corpus was generated by the model, and then attempt to evaluate the following expression, where E denotes the expected value operator:

$$T_G = \arg\max_T E(\ 1 \text{ if } L = N_C) \qquad (1)$$

This is true of the Labelled Tree Algorithm and stochastic versions of Earley's Algorithm (Stolcke, 1993), and variations such as those used in Picky parsing (Magerman and Weir, 1992). Even in probabilistic models not closely related to PCFGs, such as Spatter parsing (Magerman, 1994), expression (1) is still computed. One notable exception is Brill's Transformation-Based Error Driven system (Brill, 1993), which induces a set of transformations designed to maximize the Consistent Brackets Recall Rate. However, Brill's system is not probabilistic. Intuitively, if one were to match the parsing algorithm to the evaluation criterion, better performance should be achieved.

Ideally, one might try to directly maximize the most commonly used evaluation criteria, such as Consistent Brackets Recall (Crossing Brackets) Rate. Unfortunately, this criterion is relatively difficult to maximize, since it is time-consuming to compute the probability that a particular constituent crosses some constituent in the correct parse. On the other hand, the Bracketed Recall and Bracketed Tree Rates are easier to handle, since computing the probability that a bracket matches one in the correct parse is inexpensive. It is plausible that algorithms which optimize these closely related criteria will do well on the analogous Consistent Brackets criteria.

2.4 Which Metrics to Use

When building an actual system, one should use the metric most appropriate for the problem. For instance, if one were creating a database query system, such as an ATIS system, then the Labelled Tree (Viterbi) metric would be most appropriate. A single error in the syntactic representation of a query will likely result in an error in the semantic representation, and therefore in an incorrect database query, leading to an incorrect result. For instance, if the user request "Find me all flights on Tuesday" is misparsed with the prepositional phrase attached to the verb, then the system might wait until Tuesday before responding: a single error leads to completely incorrect behavior. Thus, the Labelled Tree criterion is appropriate.

On the other hand, consider a machine assisted translation system, in which the system provides translations, and then a fluent human manually edits them. Imagine that the system is given the foreign language equivalent of "His credentials are nothing which should be laughed at," and makes the single mistake of attaching the relative clause at the sentential level, translating the sentence as "His credentials are nothing, which should make you laugh." While the human translator must make some changes, he certainly needs to do less editing than he would if the sentence were completely misparsed. The more errors there are, the more editing the human translator needs to do. Thus, a criterion such as the Labelled Recall criterion is appropriate for this task, where the number of incorrect constituents correlates to application performance.

3 Labelled Recall Parsing

Consider writing a parser for a domain such as machine assisted translation. One could use the Labelled Tree Algorithm, which would maximize the expected number of exactly correct parses. However, since the number of correct constituents is a better measure of application performance for this domain than the number of correct trees, perhaps one should use an algorithm which maximizes the Labelled Recall criterion, rather than the Labelled Tree criterion.

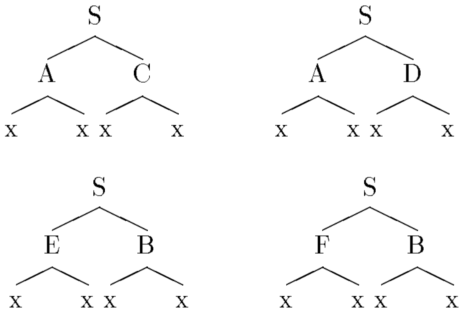
The Labelled Recall Algorithm finds that tree T_G which has the highest expected value for the Labelled Recall Rate, L/N_C (where L is the number of correct labelled constituents, and N_C is the number of nodes in the correct parse). This can be written as follows:

$$T_G = \arg\max_T E(L/N_C) \qquad (2)$$

It is not immediately obvious that the maximization of expression (2) is in fact different from the maximization of expression (1), but a simple example illustrates the difference. The following grammar generates four trees with equal probability:

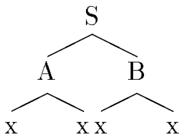
$$\begin{array}{rcll} \text{S} & \rightarrow & \text{A C} & 0.25 \\ \text{S} & \rightarrow & \text{A D} & 0.25 \\ \text{S} & \rightarrow & \text{E B} & 0.25 \\ \text{S} & \rightarrow & \text{F B} & 0.25 \\ \text{A, B, C, D, E, F} & \rightarrow & \text{x x} & 1.0 \end{array} \qquad (3)$$

The four trees are



For the first tree, the probabilities of being correct are S: 100%; A:50%; and C: 25%. Similar counting holds for the other three. Thus, the expected value of L for any of these trees is 1.75.

On the other hand, the optimal Labelled Recall parse is



This tree has 0 probability according to the grammar, and thus is non-optimal according to the Labelled Tree Rate criterion. However, for this tree the probabilities of each node being correct are S: 100%; A: 50%; and B: 50%. The expected value of L is 2.0, the highest of any tree. This tree therefore optimizes the Labelled Recall Rate.

3.1 Algorithm

We now derive an algorithm for finding the parse that maximizes the expected Labelled Recall Rate. We do this by expanding expression (2) out into a probabilistic form, converting this into a recursive equation, and finally creating an equivalent dynamic programming algorithm.

We begin by rewriting expression (2), expanding out the expected value operator, and removing the $\frac{1}{N_C}$, which is the same for all T_G, and so plays no role in the maximization.

$$T_G = \arg\max_T \sum_{T_C} P(T_C \mid w_1^n)\ |T \cap T_C| \qquad (4)$$

This can be further expanded to

$$T_G = \arg\max_T \sum_{T_C} P(T_C \mid w_1^n) \sum_{(s,t,X)\in T} 1 \text{ if } (s,t,X) \in T_C \qquad (5)$$

Now, given a PCFG with start symbol S, the following equality holds:

$$P(S \stackrel{*}{\Rightarrow} w_1^{s-1} X w_{t+1}^n \mid w_1^n) =$$

$$\sum_{T_C} P(T_C \mid w_1^n)(\ 1 \text{ if } (s,t,X) \in T_C) \qquad (6)$$

By rearranging the summation in expression (5) and then substituting this equality, we get

$$T_G = \arg\max_T \sum_{(s,t,X)\in T} P(S \stackrel{*}{\Rightarrow} w_1^{s-1} X w_{t+1}^n \mid w_1^n) \qquad (7)$$

At this point, it is useful to introduce the Inside and Outside probabilities, due to Baker (1979), and explained by Lari and Young (1990). The Inside probability is defined as $e(s,t,X) = P(X \stackrel{*}{\Rightarrow} w_s^t)$ and the Outside probability is $f(s,t,X) = P(S \stackrel{*}{\Rightarrow} w_1^{s-1} X w_{t+1}^n)$. Note that while Baker and others have used these probabilites for inducing grammars, here they are used only for parsing.

Let us define a new function, $g(s,t,X)$.

$$\begin{aligned} g(s,t,X) &= P(S \stackrel{*}{\Rightarrow} w_1^{s-1} X w_{t+1}^n \mid w_1^n) \\ &= \frac{P(S \stackrel{*}{\Rightarrow} w_1^{s-1} X w_{t+1}^n) P(X \stackrel{*}{\Rightarrow} w_s^t)}{P(S \stackrel{*}{\Rightarrow} w_1^n)} \\ &= f(s,t,X) \times e(s,t,X)/e(1,n,S) \end{aligned}$$

Now, the definition of a Labelled Recall Parse can be rewritten as

$$T_G = \arg\max_T \sum_{(s,t,X)\in T} g(s,t,X) \qquad (8)$$

Given the matrix $g(s,t,X)$, it is a simple matter of dynamic programming to determine the parse that maximizes the Labelled Recall criterion. Define

$$\begin{aligned} \mathrm{MAXC}(s,t) = {} & \max_X g(s,t,X) + \\ & \max_{r \mid s \le r < t} (\mathrm{MAXC}(s,r) + \mathrm{MAXC}(r+1,t)) \end{aligned}$$

It is clear that $\mathrm{MAXC}(1,n)$ contains the score of the best parse according to the Labelled Recall criterion. This equation can be converted into the dynamic programming algorithm shown in Figure 1.

For a grammar with r rules and k nonterminals, the run time of this algorithm is $O(n^3 + kn^2)$ since there are two layers of outer loops, each with run time at most n, and an inner loop, over nonterminals and n. However, this is dominated by the computation of the Inside and Outside probabilities, which takes time $O(rn^3)$.

By modifying the algorithm slightly to record the actual split used at each node, we can recover the

```
for length := 2 to n
   for s := 1 to n-length+1
      t := s + length - 1;
      loop over nonterminals X
         let max_g:=maximum of g(s,t,X)
      loop over r such that s <= r < t
         let best_split:=
            max of maxc[s,r] + maxc[r+1,t]
      maxc[s, t] := max_g + best_split;
```

Figure 1: Labelled Recall Algorithm

best parse. The entry `maxc[1, n]` contains the expected number of correct constituents, given the model.

4 Bracketed Recall Parsing

The Labelled Recall Algorithm maximizes the expected number of correct labelled constituents. However, many commonly used evaluation metrics, such as the Consistent Brackets Recall Rate, ignore labels. Similarly, some grammar induction algorithms, such as those used by Pereira and Schabes (1992) do not produce meaningful labels. In particular, the Pereira and Schabes method induces a grammar from the brackets in the treebank, ignoring the labels. While the induced grammar has labels, they are not related to those in the treebank. Thus, although the Labelled Recall Algorithm could be used in these domains, perhaps maximizing a criterion that is more closely tied to the domain will produce better results. Ideally, we would maximize the Consistent Brackets Recall Rate directly. However, since it is time-consuming to deal with Consistent Brackets, we instead use the closely related Bracketed Recall Rate.

For the Bracketed Recall Algorithm, we find the parse that maximizes the expected Bracketed Recall Rate, B/N_C. (Remember that B is the number of brackets that are correct, and N_C is the number of constituents in the correct parse.)

$$T_G = \arg\max_T E(B/N_C) \tag{9}$$

Following a derivation similar to that used for the Labelled Recall Algorithm, we can rewrite equation (9) as

$$T_G = \arg\max_T \sum_{(s,t)\in T} \sum_X P(S \stackrel{*}{\Rightarrow} w_1^{s-1} X w_{t+1}^n \mid w_1^n) \tag{10}$$

The algorithm for Bracketed Recall parsing is extremely similar to that for Labelled Recall parsing. The only required change is that we sum over the symbols X to calculate `max_g`, rather than maximize over them.

5 Experimental Results

We describe two experiments for testing these algorithms. The first uses a grammar without meaningful nonterminal symbols, and compares the Bracketed Recall Algorithm to the traditional Labelled Tree (Viterbi) Algorithm. The second uses a grammar with meaningful nonterminal symbols and performs a three-way comparison between the Labelled Recall, Bracketed Recall, and Labelled Tree Algorithms. These experiments show that use of an algorithm matched appropriately to the evaluation criterion can lead to as much as a 10% reduction in error rate.

In both experiments the grammars could not parse some sentences, 0.5% and 9%, respectively. The unparsable data were assigned a right branching structure with their rightmost element attached high. Since all three algorithms fail on the same sentences, all algorithms were affected equally.

5.1 Experiment with Grammar Induced by Pereira and Schabes Method

The experiment of Pereira and Schabes (1992) was duplicated. In that experiment, a grammar was trained from a bracketed form of the TI section of the ATIS corpus[1] using a modified form of the Inside-Outside Algorithm. Pereira and Schabes then used the Labelled Tree Algorithm to select the best parse for sentences in held out test data. The experiment was repeated here, except that both the Labelled Tree and Labelled Recall Algorithm were run for each sentence. In contrast to previous research, we repeated the experiment ten times, with different training set, test set, and initial conditions each time.

Table 1 shows the results of running this experiment, giving the minimum, maximum, mean, and standard deviation for three criteria, Consistent Brackets Recall, Consistent Brackets Tree, and Bracketed Recall. We also display these statistics for the paired differences between the algorithms. The only statistically significant difference is that for Consistent Brackets Recall Rate, which was significant to the 2% significance level (paired t-test).

[1] For our experiments the corpus was slightly cleaned up. A diff file for "ed" between the original ATIS data and the cleaned-up version is available from ftp://ftp.das.harvard.edu/pub/goodman/atis-ed/ ti_tb.par-ed and ti_tb.pos-ed. The number of changes made was small, less than 0.2%

Criteria	Min	Max	Mean	SDev
Labelled Tree Algorithm				
Cons Brack Rec	86.06	93.27	90.13	2.57
Cons Brack Tree	51.14	77.27	63.98	7.96
Brack Rec	71.38	81.88	75.87	3.18
Bracketed Recall Algorithm				
Cons Brack Rec	88.02	94.34	91.14	2.22
Cons Brack Tree	53.41	76.14	63.64	7.82
Brack Rec	72.15	80.69	76.03	3.14
Differences				
Cons Brack Rec	-1.55	2.45	1.01	1.07
Cons Brack Tree	-3.41	3.41	-0.34	2.34
Brack Rec	-1.34	2.02	0.17	1.20

Table 1: Percentages Correct for Labelled Tree versus Bracketed Recall for Pereira and Schabes

Thus, use of the Bracketed Recall Algorithm leads to a 10% reduction in error rate.

In addition, the performance of the Bracketed Recall Algorithm was also qualitatively more appealing. Figure 2 shows typical results. Notice that the Bracketed Recall Algorithm's Consistent Brackets Rate (versus iteration) is smoother and more nearly monotonic than the Labelled Tree Algorithm's. The Bracketed Recall Algorithm also gets off to a much faster start, and is generally (although not always) above the Labelled Tree level. For the Labelled Tree Rate, the two are usually very comparable.

5.2 Experiment with Grammar Induced by Counting

The replication of the Pereira and Schabes experiment was useful for testing the Bracketed Recall Algorithm. However, since that experiment induces a grammar with nonterminals not comparable to those in the training, a different experiment is needed to evaluate the Labelled Recall Algorithm, one in which the nonterminals in the induced grammar are the same as the nonterminals in the test set.

5.2.1 Grammar Induction by Counting

For this experiment, a very simple grammar was induced by counting, using a portion of the Penn Tree Bank, version 0.5. In particular, the trees were first made binary branching by removing epsilon productions, collapsing singleton productions, and converting n-ary productions ($n > 2$) as in figure 3. The resulting trees were treated as the "Correct" trees in the evaluation. Only trees with forty or fewer symbols were used in this experiment.

A grammar was then induced in a straightforward way from these trees, simply by giving one count for each observed production. No smoothing was done. There were 1805 sentences and 38610 nonterminals in the test data.

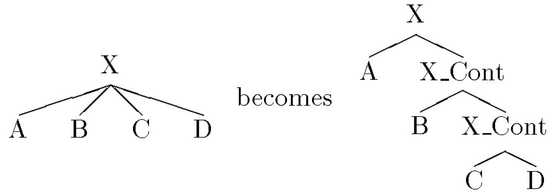


Figure 3: Conversion of Productions to Binary Branching

	Recall	Tree
Brackets	Bracketed Recall	(NP-Complete)
Labels	Labelled Recall	Labelled Tree

Table 3: Metrics and Corresponding Algorithms

5.2.2 Results

Table 2 shows the results of running all three algorithms, evaluating against five criteria. Notice that for each algorithm, for the criterion that it optimizes it is the best algorithm. That is, the Labelled Tree Algorithm is the best for the Labelled Tree Rate, the Labelled Recall Algorithm is the best for the Labelled Recall Rate, and the Bracketed Recall Algorithm is the best for the Bracketed Recall Rate.

6 Conclusions and Future Work

Matching parsing algorithms to evaluation criteria is a powerful technique that can be used to improve performance. In particular, the Labelled Recall Algorithm can improve performance versus the Labelled Tree Algorithm on the Consistent Brackets, Labelled Recall, and Bracketed Recall criteria. Similarly, the Bracketed Recall Algorithm improves performance (versus Labelled Tree) on Consistent Brackets and Bracketed Recall criteria. Thus, these algorithms improve performance not only on the measures that they were designed for, but also on related criteria.

Furthermore, in some cases these techniques can make parsing fast when it was previously impractical. We have used the technique outlined in this paper in other work (Goodman, 1996) to efficiently parse the DOP model; in that model, the only previously known algorithm which summed over all the possible derivations was a slow Monte Carlo algorithm (Bod, 1993). However, by maximizing the

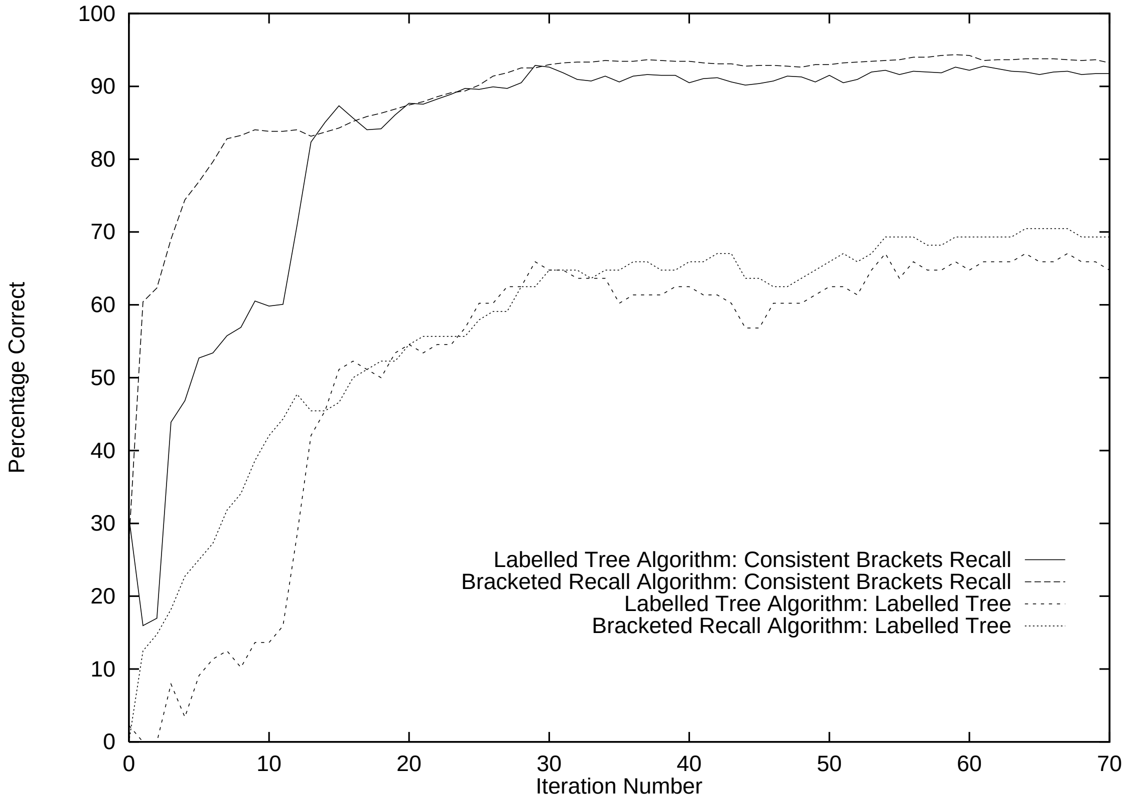


Figure 2: Labelled Tree versus Bracketed Recall in Pereira and Schabes Grammar

	Criterion				
Algorithm	Label Tree	Label Recall	Brack Recall	Cons Brack Recall	Cons Brack Tree
Label Tree	*4.54%*	48.60%	60.98%	66.35%	12.07%
Label Recall	3.71%	*49.66%*	61.34%	68.39%	11.63%
Bracket Recall	0.11%	4.51%	*61.63%*	68.17%	11.19%

Table 2: Grammar Induced by Counting: Three Algorithms Evaluated on Five Criteria

Labelled Recall criterion, rather than the Labelled Tree criterion, it was possible to use a much simpler algorithm, a variation on the Labelled Recall Algorithm. Using this technique, along with other optimizations, we achieved a 500 times speedup.

In future work we will show the surprising result that the last element of Table 3, maximizing the Bracketed Tree criterion, equivalent to maximizing performance on Consistent Brackets Tree (Zero Crossing Brackets) Rate in the binary branching case, is NP-complete. Furthermore, we will show that the two algorithms presented, the Labelled Recall Algorithm and the Bracketed Recall Algorithm, are both special cases of a more general algorithm, the General Recall Algorithm. Finally, we hope to extend this work to the n-ary branching case.

7 Acknowledgements

I would like to acknowledge support from National Science Foundation Grant IRI-9350192, National Science Foundation infrastructure grant CDA 94-01024, and a National Science Foundation Graduate Student Fellowship. I would also like to thank Stanley Chen, Andrew Kehler, Lillian Lee, and Stuart Shieber for helpful discussions, and comments on earlier drafts, and the anonymous reviewers for their comments.

References

[Baker1979] Baker, J.K. 1979. Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550, Boston, MA, June.

[Bod1993] Bod, Rens. 1993. Using an annotated corpus as a stochastic grammar. In *Proceedings of the Sixth Conference of the European Chapter of the ACL*, pages 37–44.

[Brill1993] Brill, Eric. 1993. *A Corpus-Based Approach to Language Learning.* Ph.D. thesis, University of Pennsylvania.

[Goodman1996] Goodman, Joshua. 1996. Efficient algorithms for parsing the DOP model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* To appear.

[Lari and Young1990] Lari, K. and S.J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

[Magerman1994] Magerman, David. 1994. *Natural Language Parsing as Statistical Pattern Recognition.* Ph.D. thesis, Stanford University University, February.

[Magerman and Weir1992] Magerman, D.M. and C. Weir. 1992. Efficiency, robustness, and accuracy in picky chart parsing. In *Proceedings of the Association for Computational Linguistics.*

[Pereira and Schabes1992] Pereira, Fernando and Yves Schabes. 1992. Inside-Outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the ACL*, pages 128–135, Newark, Delaware.

[Stolcke1993] Stolcke, Andreas. 1993. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Technical Report TR-93-065, International Computer Science Institute, Berkeley, CA.