Edge-Based Best-First Chart Parsing

Sharon Goldwater

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-98-04
May 1998

Edge-Based Best-First Chart Parsing

Sharon Goldwater*

May 1, 1998

Abstract

Natural language grammars are often very large and full of ambiguities, making standard computer parsers too slow to be practical for many tasks. Best-first parsing attempts to address this problem by preferentially working to expand subparses that are judged "good" by some probabilistic figure of merit. We explain the standard non-probabilistic and best-first chart parsing paradigms, then describe a new method of best-first parsing which improves upon previous work by ranking subparses at a more fine-grained level, speeding up parsing by approximately a factor of 20 over the best previous results. Moreover, these results are achieved with a higher level of accuracy than is obtained by parsing to exhaustion.

1 Introduction

Natural language processing is a subfield of artificial intelligence which deals with trying to get computers to handle human language in a useful way. Having computers which could understand human language would simplify human-computer interaction and allow computers access to the huge amount of information available in text. In general, knowing a sentence's structure is a prerequisite to finding its meaning, which makes parsing— finding the syntactic structure of sentences— one of the basic tasks in natural language processing. Knowledge of sentence structure also has the potential to improve performance on more specialized tasks which are the focus of current research, including automatic translation, document summarization, dictation, and voice-activated interfaces.

*research in conjunction with Eugene Charniak and Mark Johnson

Unfortunately, right now many of these applications do not make use of much sentence structure, in part because parsing is often very slow. There are standard parsing algorithms (e.g. chart parsers) which are cubic in the length of the sentence, but real sentences can reach 40 or more words in length, and the huge size of many natural language grammars introduces large constant factors into the time bound. In addition, the usefulness of these parsers is questionable, since they find every possible parse of a sentence (that is, they are exhaustive), but most of these parses are nowhere close to the parse (or handful of parses) a human would propose, and a standard computer parser provides no information about which parses are better than others. Rather than producing all possible parses, the ideal computer parser would be able to produce one, or a few, of the "best" parses— those which correspond to a human's intuition.

A common approach to the latter problem is to adopt a probabilistic framework, using a probabilistic context-free grammar (PCFG) where each production rule $A \rightarrow B_0 \ldots B_n$ is labeled with a probability as to how likely nonterminal A is to expand in this way. This allows the likelihood of a complete parse to be evaluated by multiplying the probabilities of the rules which generated the parse. An exhaustive probabilistic parser is still very slow, but at least it is able to decide which parses are better than others.

The probabilistic framework can be exploited further by using it to address the problem of speed as well. Parsing is essentially a search problem with the goal of finding a good parse (or the best parse) in the space of all possible parses. As such, we can build a parser which uses statistics to guide the search, thus speeding up the parsing process. In "Best-first" probabilistic parsing, introduced by Bobrow [1] and Chitrao and Grishman [5], partial parses are ranked according to some probabilistic figure of merit (FOM), and higher ranked subparses are given priority in the search— that is, the parser preferentially works to expand subparses with higher FOMs.

Naturally, the speed of a best-first parser and the correctness of its results depend on the FOM used to rank constituents. Several FOMs have been proposed for this purpose [12, 9, 10]. One of the more extensive comparisons is presented by Caraballo and Charniak [2] (referred to henceforth as C&C). Our work relies heavily on C&C's results, which are described in section 3. Recent work by Goodman [7] uses an FOM which is similar to C&C's, but probably more accurate. His FOM requires a beam search for parsing, and thus cannot be applied here, but we will briefly compare our results to his in section 5.

2 Standard Chart Parsing

Before elaborating on the idea of best-first parsing and describing our research, we will first explain the basic chart parsing algorithm on which our work is based. Chart parsing is a dynamic programming algorithm which can be used to find a possibly exponential number of different parses in $O(n^3)$ time by reusing information stored in a *chart*. The chart is an n by n array whose entries are terminal and nonterminal symbols in the grammar (constituents). Each entry is indexed by its start position in the sentence and its length. For example, if cell [3,2] contains the symbol NP, this means the parser has found a two-word-long noun phrase starting with the third word in the sentence.

A chart parser has two other major components: the *agenda*, a list or stack of items which are waiting to be entered into the chart, and a set of *edges*. An edge is a rule which can be applied to chart entries, combining them to form new entries. Each edge keeps track of which grammar rule it corresponds to, which of the constituents on the right hand side of that rule have been found already, and which constituents must still be found in order to complete the left hand side constituent. For example, the edge

$$\text{VP} \rightarrow \text{VP NP} \circ \text{PP PP}$$

indicates that the parser has found a verb phrase and a noun phrase, and will complete a larger verb phrase if it finds two prepositional phrases following the noun phrase. The edge

$$\text{VP} \rightarrow \text{VP NP PP PP} \circ$$

indicates that the parser has found all the constituents on the right hand side of the rule, and therefore has completed the VP on the left.

For each edge, the parser also stores its starting point (the starting position of the first constituent on the right side of its rule), its ending point (the ending position of the last constituent before the $\circ$), and a list of the actual constituents in the chart which correspond to the right hand side of the rule. This list is used when we wish to reconstruct the parse tree of the sentence. Each completed constituent also stores such a list.

The basic chart parsing algorithm is shown in Figure 1. For a full simulation of the algorithm on an example sentence, see section 1.4 of [3].

Figure 1: The Chart Parsing Algorithm

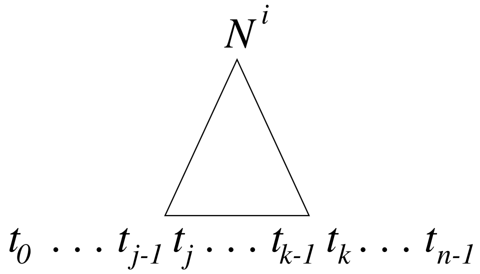
Loop until the agenda is empty:

1. Remove a constituent, A, from the agenda. Let i be A's type.
2. If a constituent, B, also of type i and with the same start and end points as A, is already in the chart, append the edge list of A to the edge list of B, and return to step 1. (We have just found a new parse for this constituent)
3. Add A to the chart.
4. For each grammar rule where the first symbol on the right hand side is of type i, add an edge for that rule to the chart.
5. For each edge in the chart which has a symbol of type i immediately following the $\circ$, create an extended edge and add it to the chart.
6. For each newly completed edge (if any), add a constituent to the agenda with the appropriate type, start and end points, and edge list.

To extend an edge e with constituent A:

1. Create a new edge e', where e''s rule looks like e's rule, but with the $\circ$ shifted one place to the right.
2. Set the start position of e' to the start position of e.
3. Set the end position of e' to the end position of A.
4. Set the list of right-hand-side constituents of e' to that of e, with the addition of A.

Figure 2: $N^i_{j,k}$



3 Constituent-based Best-first Chart Parsing

It is easy to modify the standard chart parser to implement a best-first approach, simply by replacing the agenda list with a priority queue sorted by the chosen figure of merit. Constituents with higher FOMs are now popped off the agenda and added to the chart before those with lower FOMs. After the parser has found a full parse (or a small number of parses), the algorithm simply terminates, and all items remaining on the agenda are discarded.

This approach is exactly the one adopted by C&C, whose paper describes a series of experiments with a best-first probabilistic chart parser using various figures of merit. We call their work "constituent-based" because their FOMs are used to rank completed constituents only. Incomplete edges are always extended as soon as an appropriate constituent is popped off the agenda and added to the chart. Note that in both their work and ours, the parser is not given the actual words in the sentence as input, and uses only their tags (parts of speech).

C&C begin by proposing an "ideal" FOM to use for ranking each constituent:

$$p(N^i_{j,k} \mid t_{0,n}) \tag{1}$$

where $t_{0,n}$ is the sequence of tags t_0 through t_{n-1} which make up the sentence, and $N^i_{j,k}$ is a nonterminal of type i which roots the subtree whose leaves are the tags $t_j, \ldots, t_{k-1}$ (see Figure 2). So the merit of a particular constituent corresponds to its probability given the entire sentence.

Unfortunately, the above FOM is not computable until the sentence has been completely parsed, so the remainder of C&C is devoted to evaluating various approximations to $p(N^i_{j,k} \mid t_{0,n})$. The approximation which yields their best results is based on the equation

$$p(N^i_{j,k} \mid t_{0,n}) \approx \frac{p(N^i_{j,k} \mid t_{j-1})\beta(N^i_{j,k})p(t_k \mid N^i_{j,k})}{p(t_{j,k+1} \mid t_{j-1})} \tag{2}$$

where $\beta(N^i_{j,k})$, the *inside probability* of constituent N^i, is defined to be the probability that N^i generates tags $t_j \ldots t_{k-1}$:

$$\beta(N^i_{j,k}) = p(t_{j,k} \mid N^i_{j,k}). \tag{3}$$

Essentially, Equation 2 approximates $p(N^i_{j,k} \mid t_{0,n})$ by using $\beta(N^i_{j,k})$ to give the probability of $N^i_{j,k}$ in isolation, and $p(N^i_{j,k} \mid t_{j-1})$ and $p(t_k \mid N^i_{j,k})$ to approximate the effects of its context in the sentence.

The derivation of Equation 2 is as follows:

Using the definition of conditional probability,

$$p(N^i_{j,k} \mid t_{0,n}) = \frac{p(N^i_{j,k}, t_{0,n})}{p(t_{0,n})}$$

Since $p(t_{0,n})$ is just the joint probability of all the tags, we can rewrite it as $p(t_{0,j}, t_{j,k}, t_k, t_{k+1,n})$. We then apply the chain rule and cancel $p(t_{0,j})$:

$$\begin{aligned}
&p(N^i_{j,k} \mid t_{0,n}) \\
&= \frac{p(N^i_{j,k}, t_{0,j}, t_{j,k}, t_k, t_{k+1,n})}{p(t_{0,j}, t_{j,k}, t_k, t_{k+1,n})} \\
&= \frac{p(t_{0,j})p(N^i_{j,k} \mid t_{0,j})p(t_{j,k} \mid N^i_{j,k}, t_{0,j})p(t_k \mid N^i_{j,k}, t_{0,j}, t_{j,k})p(t_{k+1,n} \mid N^i_{j,k}, t_{0,j}, t_{j,k}, t_k)}{p(t_{0,j})p(t_{j,k+1} \mid t_{0,j})p(t_{k+1,n} \mid t_{0,k+1})} \\
&= \frac{p(N^i_{j,k} \mid t_{0,j})p(t_{j,k} \mid N^i_{j,k}, t_{0,j})p(t_k \mid N^i_{j,k}, t_{0,k})p(t_{k+1,n} \mid N^i_{j,k}, t_{0,k+1})}{p(t_{j,k+1} \mid t_{0,j})p(t_{k+1,n} \mid t_{0,k+1})}.
\end{aligned}$$

To simplify the equation, we assume that the sequence of tags generated by a nonterminal depends only on that nonterminal, and that $t_{k+1,n}$ are independent of $N^i_{j,k}$ given $t_{0,k+1}$. We now have:

$$\begin{aligned}
p(N^i_{j,k} \mid t_{0,n}) &\approx \frac{p(N^i_{j,k} \mid t_{0,j})p(t_{j,k} \mid N^i_{j,k})p(t_k \mid N^i_{j,k}, t_{0,k})p(t_{k+1,n} \mid t_{0,k+1})}{p(t_{j,k+1} \mid t_{0,j})p(t_{k+1,n} \mid t_{0,k+1})} \\
&= \frac{p(N^i_{j,k} \mid t_{0,j})\beta(N^i_{j,k})p(t_k \mid N^i_{j,k}, t_{0,k})}{p(t_{j,k+1} \mid t_{0,j})}.
\end{aligned}$$

We now assume that $p(N^i_{j,k})$ depends only on the probability of the preceding tag, and similarly, that the probability of the following tag depends only on $p(N^i_{j,k})$. In addition, we simplify the denominator by disregarding the effects of $t_{0,j-1}$, so $p(t_{j,k+1} \mid t_{0,j}) \approx p(t_{j,k+1} \mid t_{j-1})$. This gives us the approximation in Equation 2.

We approximate the joint probability in the denominator of Equation 2 with a bitag model— that is, we assume each tag depends only on the previous tag:

$$p(t_{j,k+1}) \approx \prod_{i=j}^{k} p(t_i \mid t_{i-1}). \tag{4}$$

It is now easy to calculate the denominator by simply collecting the bitag probabilities from the training data and multiplying them. The "boundary statistics" $p(N^i_{j,k} \mid t_{j-1})$ and $p(t_k \mid N^i_{j,k})$ can also be collected from the training data, as described in C&C.

As for the inside probabilities, the value of $\beta(N^i_{j,k})$ is given by

$$\beta(N^i_{j,k}) = \sum_{e \in \mathcal{E}^i_{j,k}} \beta(e) \tag{5}$$

where $\mathcal{E}^i_{j,k}$is the set of completed edges which can be used to create $N^i_{j,k}$, and $\beta(e)$ is the probability of the grammar rule r_e associated with e times the product of the inside probabilities of all the constituents on the right hand side of r_e:

$$\beta(e) = p(r_e) \prod_{X^s_{t,u} \in \mathrm{rhs}(r_e)} \beta(X^s_{t,u}) \tag{6}$$

We can't compute this number exactly without parsing to exhaustion to find every member of $\mathcal{E}^i_{j,k}$, but we can compute an estimate online, updating it as new constituents are added to the chart. We are given a sequence of tags as input, so we begin by setting β to 1 for each tag. As each new constituent $N^i_{j,k}$ is formed, we set its β value to $\beta(e)$, where e is the rule expansion used to create $N^i_{j,k}$. If, at some later time, a more probable parse for $N^i_{j,k}$is found using rule e', we update β to be $\beta(e')$. Thus, $\beta(N^i_{j,k})$ is always approximated by

$$\max_{e \in \mathcal{E}^i_{j,k}} \beta(e). \tag{7}$$

This estimate differs from that of C&C, who estimated $\beta(N^i_{j,k})$ by summing the probabilities of all the parses found so far for $N^i_{j,k}$. Our estimate, though not as accurate, is simpler to update, and it seems reasonable to use an estimate based on the most probable parse, since this is the parse we are searching for.

Given our methods for collecting the statistics used in the FOM, a problem arises. We collect statistics on the probabilities of tag sequences in two different ways. In the numerator, the tag sequence probabilities used in β come from the PCFG model, whereas in the denominator, the probabilities are generated by a bitag model. This introduces some discrepancy between the numerator and the denominator of the fraction. In particular, the bitag model tends to assign higher probability to a sequence of tags than does the PCFG model. This fact is not mentioned in C&C's paper, although they noted it at the time.

We have found experimentally in our training data that the bigram model probabilities are approximately 1.3 times as high as the PCFG probabilities. For single tags, this is a fairly small discrepancy, but as the sequence of tags becomes longer, the denominator becomes increasingly large in comparison to the numerator, thus devaluing larger constituents and delaying the parser from finding a full parse. In order to correct this problem, we assign a probability of $\eta > 1$ to each known tag in the PCFG model, rather than a probability of 1. This has the effect of multiplying $\beta(N^i_{j,k})$ by η^{k-j}. In section 5 we show results for η between 1.0 and 2.4.

4 Conversion to Edge-based Parsing

We extend the work of C&C by using their FOM to rank not only complete constituents, but edges as well. A simple way to implement this extension is by modifying the grammar to obtain an equivalent PCFG containing only binary and unary rules. By equivalent, we mean that the new grammar, G_1, produces exactly the same strings with the same probabilities as the old grammar, G_0. Given a grammar with only binary and unary rules, we no longer need to use incomplete edges at all. Anytime we add a constituent to the chart, we need only look to its immediate left and right for constituents to combine with and put any new constituents formed on the agenda. (This is essentially the CKY parsing algorithm, modified to handle unary rules.)

To obtain G_1 from G_0, we apply the technique of *left-factoring*, introduced by Hopcroft and Ulman [8]. In the original grammar, we have rules of the form $A \to B_0 \ldots B_n \ : \ q$, where q is the probability of the production $A \to B_0 \ldots B_n$. For any rule of this type where $n \geq 2$, we create a new set of binary rules:

$$\begin{array}{rcll} A & \rightarrow & \langle B_0 \ldots B_{n-1} \rangle B_n & : q \\ \langle B_0 \ldots B_{n-1} \rangle & \rightarrow & \langle B_0 \ldots B_{n-2} \rangle B_{n-1} & : 1.0 \\ \vdots & & & \\ \langle B_0 B_1 \rangle & \rightarrow & B_0 B_1 & : 1.0 \end{array}$$

Here, each of the $\langle B_0 \ldots B_i \rangle$ is a new nonterminal symbol in G_1 which corresponds to the incomplete edge $A \rightarrow B_0 \ldots B_i \circ B_{i+1} \ldots B_n$ in G_0.

As an example, the rule

$$\text{VP} \rightarrow \text{V NP PP PP} : .004$$

becomes

$$\begin{array}{rcll} \text{VP} & \rightarrow & \langle \text{VP NP PP} \rangle \text{ PP} & : .004 \\ \langle \text{VP NP PP} \rangle & \rightarrow & \langle \text{VP NP} \rangle \text{ PP} & : 1.0 \\ \langle \text{VP NP} \rangle & \rightarrow & \text{VP NP} & : 1.0 \end{array}$$

and the new constituent $\langle \text{V NP PP} \rangle$ is equivalent to the old edge VP $\rightarrow$ V NP PP $\circ$ PP.

By assigning a probability of 1 to all the rules with $\langle B_0 \ldots B_i \rangle$ on the left, we ensure that the probability p_0 of expanding A into a tree with leaves $B_0 \ldots B_n$ using G_0 is the same as the probability p_1 of the same expansion in G_1. Since the probability of a tree is the product of the probability of the rules used to produce that tree, we have

$$\begin{array}{rcl} p_0 & = & p(A \rightarrow B_0 \ldots B_n) \\ & = & q \end{array}$$

and

$$\begin{array}{rcl} p_1 & = & p(A \rightarrow \langle B_0 \ldots B_{n-1} \rangle B_n) p(\langle B_0 \ldots B_{n-1} \rangle \rightarrow \langle B_0 \ldots B_{n-2} \rangle B_{n-1}) \ldots p(\langle B_0 B_1 \rangle \rightarrow B_0 B_1) \\ & = & q(1.0) \ldots (1.0) \\ & = & q \end{array}$$

Now suppose we have a full parse tree T_0 obtained from G_0. The corresponding tree T_1 from G_1 will have the same probability, since each single rule expansion in T_0 has a corresponding subtree in T_1 with the same probability. Multiplying the probabilities of all these subtrees will yield the same result as multiplying all the rule expansion probabilities in T_0. Finally, since the probability of a string is the sum of the probabilities of all possible parses of that string, and we know that each parse has the same probability in each grammar, we know that G_0 and G_1 are in fact equivalent.

Once we have G_1, switching to an edge-based best-first parser is simple. In constituent-based parsing, edges are stored in the chart, and when we add a constituent to the chart, we extend all possible edges immediately. Now, when we add a constituent to the chart, we combine it with items on either side (if possible), but instead of leaving the new items in the chart, we rank them and place them on the agenda. By using the FOM on lower-level items, the parser can rule out bad constituents even before completing them, enabling it to concentrate on better constituents and reach a parse faster.

5 Experimental Results

For our experiment, we used a tree-bank grammar induced from sections 2 - 21 of the Penn Wall Street Journal text [11], with section 22 reserved for testing. All sentences of length greater than 40 were ignored for testing purposes as done in both C&C and Goodman [7]. We applied the left-factoring technique described above to the grammar.

The accuracy of our parser is reported in terms of *precision* and *recall*. The precision of a parsed sentence is the number of constituents the parser identified correctly (correct start and end points, and correct label) divided by the number of constituents in the hand-parsed version. Recall is the number of constituents identified correctly divided by the total number identified. The trees in the treebank tend to be a bit flatter than those produced by our parser, so the average precision over the test corpus is generally slightly higher than the average recall. However, the difference is fairly small, so for simplicity we will sometimes combine precision and recall into a single measure. Henceforth, the term *accuracy* will refer to the quantity (precision + recall)/2.

We chose to measure the amount of work done by the parser in terms of the average number of edges popped off the agenda before finding a parse. This method has the advantage of being platform independent, as well as providing a measure of "perfection". Here, perfection is the minimum number of edges we would need to pop off the agenda in order to create the correct parse. For the left-factored grammar, where each popped edge is a completed constituent, this number is simply the number of terminals plus nonterminals in the sentence— on average, 47.5. (At the other extreme, ex-

Figure 3: η vs. Popped Edges

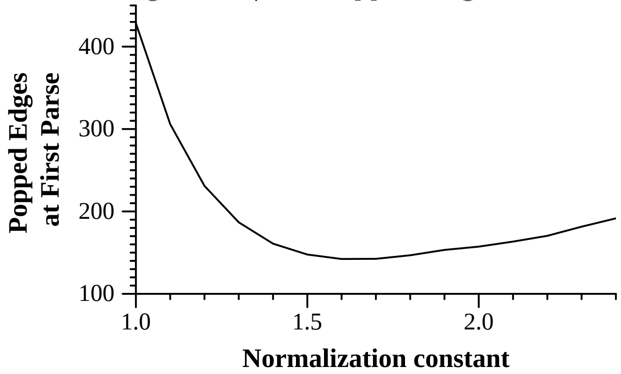
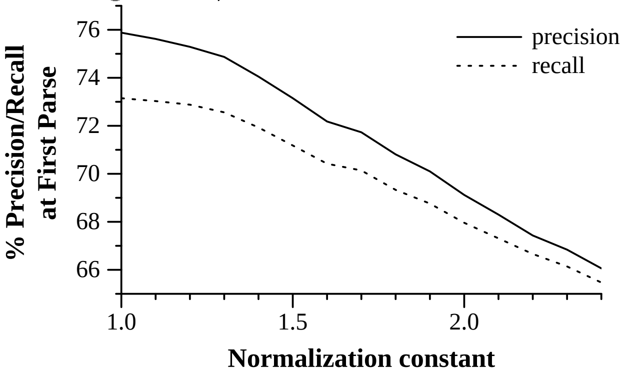


Figure 4: η vs. Precision and Recall



haustive parsing of these sentences requires an average of about 1.2 million edges per sentence.)

Our algorithm includes some measures to reduce the number of items on the agenda, and thus (presumably) the number of popped edges. Each time we add a constituent to the chart, we combine it with the constituents on either side of it, potentially creating several new edges. For each of these new edges, we check to see if a matching constituent (i.e. a constituent with the same head, start, and end points) already exists in either the agenda or the chart. If there is no match, we simply add the new edge to the agenda. If there is a match but the old parse of $N^i_{j,k}$ is better than the new one, we discard the new parse. Finally, if we have found a better parse of $N^i_{j,k}$, we add the new edge to the agenda, removing the old one if it has not already been popped.

We tested the parser on section section 22 of the WSJ text with various normalization constants η, working on each sentence only until we reached the first full parse. For each sentence we recorded the number of popped edges needed to reach the first parse, and the precision and recall of that parse. The average number of popped edges to first parse as a function of η is shown in Figure 3, and the average precision and recall are shown in Figure 4.

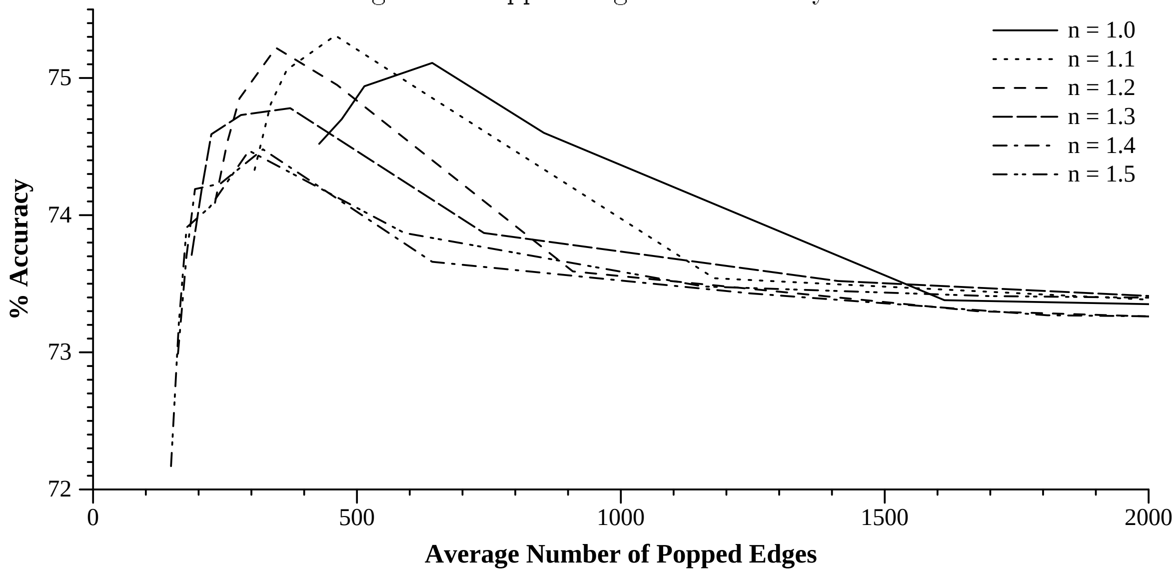
In these two figures, we see that the number of popped edges decreases as η increases from 1.0 to 1.7, then begins to increase again, while the precision and recall decrease steadily as η increases. (Note that, because we used a left-factored grammar for parsing, the trees produced by the parser are structured and labeled differently from those in the treebank. In order to calculate precision and recall, we applied a simple tree transform to the parser's output to reverse the left-factoring, then calculated the figures as usual.)

At first glance, Figure 3 seems at odds with our predicted optimal normalization factor, 1.3. However, as explained earlier, the higher η is, the higher merit longer constituents receive. Therefore, letting $\eta = 1.3$ balances out the numerator and denominator of our FOM, but a higher η will push the parser toward longer constituents, thus allowing it to find a full parse more quickly. The sharp drop-off in precision and recall when $\eta > 1.3$ is consistent with this hypothesis, since the parser is apparently focusing on longer constituents to the detriment of smaller constituents which are in fact more probable.

This interpretation suggests an investigation into whether we can gain accuracy with higher η by letting the parser continue past the first parse (i.e. pop more edges). To answer this question, we ran the parser again, allowing it to continue parsing until it had popped 20 times as many edges as needed to reach the first parse. The results of this experiment are shown in Figure 5, where we plot accuracy as a function of edges.

Note that the accuracy of the parse increases given extra time for all values of η, but that this effect is much more pronounced for $\eta > 1.3$. In every case, all of the increase is achieved with only 1.5 to 2 times as many edges as needed for the first parse. For η between 1.0 and 1.2, the highest accuracy is almost the same, about 75.2%, but this value is reached with an average of slightly under 400 edges when $\eta = 1.2$, compared to about 650 when $\eta = 1.0$. The predicted optimal value of 1.3 does not do quite as well in terms of highest accuracy, though it reaches 74.5% in about 200 edges. As η increases past 1.3, the first parse is reached only slightly faster and

Figure 5: Popped Edges vs. Accuracy



with much poorer accuracy, and highest accuracy drops as well.

One interesting property of these results is that as we continue to pop more edges, the accuracy begins to drop again. In fact, for η below 1.4, even the first parses score better on average than the parses obtained by exhaustive parsing, even though the exhaustive parses are strictly more probable. One possible explanation for these phenomena is that our figure of merit may cause the parser to prefer constituents which are probable given their surrounding preterminals, whereas the maximum likelihood parse does not take this information into account. Another possibility is that our FOM is, in effect, partially imitating Goodman's "Labeled Recall" parser [6], which tries to maximize labeled bracket recall with the test set, rather than returning the maximum likelihood parse.

In any case, probably the most remarkable feature of these results is the small number of edges needed to find a good parse. Our most accurate results can be achieved with only ten times as many edges as the the average number required, 47.5. And with a small sacrifice in accuracy, we are able to find a parse with as few as three times the required number.

The significance of these numbers becomes apparent when comparing our results to those achieved by Goodman [7] and C&C. It is difficult to fully compare Goodman's results to ours, since he measures processor time

Figure 6: Edges vs. Sentences Parsed

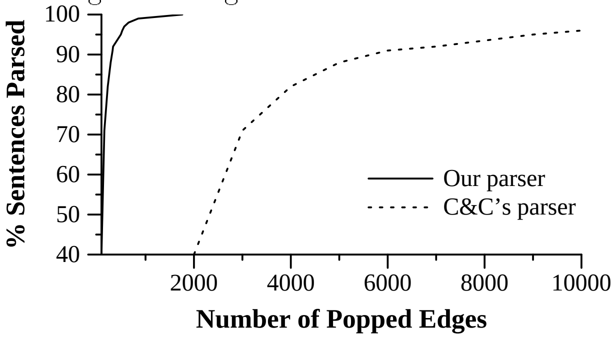


Table 1: Edges vs. Sentences Parsed

% Sentences Parsed	Our Edgecount	C&C Edgecount
40	90	2000
71	150	3000
82	220	4000
91	320	6000
95	490	9000
96	520	10000
100	1760	

rather than edges, but in one case he does give a measure of edges per second. Combining this with his figures for parsing time suggests that his parser uses tens of thousands of edges per sentence, as compared to a few hundred for our parser. C&C do give results in terms of edges, reporting the percentage of sentences of length 18-26 which have been parsed as a function of the number of edges used. We performed this same experiment with our own parser, setting $\eta = 1.2$. Both sets of results are displayed in Figure 6. It is clear from the graph that our parser uses an order of magnitude or so fewer edges than C&C's. For exact numbers, see Table 1.

6 Conclusion

The advantages of adopting an edge-based technique in best-first chart parsing using C&C's figure of merit are clear: this method reduces the number of edges needed to find a first parse by approximately a factor of 20, giving parses which use as few as three times the optimal number of edges, and produces parses which are more accurate than those produced by exhaustive parsing. Furthermore, it is easy to convert a standard best-first parser to an edge-based model by left-factoring the grammar.

As a final note, it is important to realize that although this work was done with a parser which had access only to the tags in the sentence and using a grammar induced directly from the Penn Treebank, it should be possible to apply the same techniques to best-first parsers using other types of statistics, some of which are much more accurate than the one used here. For example, Charniak's lexicalized parser [4] uses word-based statistics in addition to tag-based statistics, achieving accuracy levels in the mid 80% range. The methods used here to speed up the tag-based parser ought to work for the lexicalized parser as well, and more research in this area seems warranted.

7 Acknowledgments

I would like to thank Eugene Charniak and Mark Johnson, without whom this research would never have happened, and Eugene Charniak (again) for encouraging me to actually write about it. Thanks also to the UTRA program for funding me over the summer to work on this project, to the people in the AI Lab for making life interesting, and to Sam for his advice and forbearance.

References

[1] Robert J. Bobrow. Statistical agenda parsing. In *DARPA Speech and Language Workshop*, pages 222–224, 1990.

[2] Sharon Caraballo and Eugene Charniak. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, Forthcoming.

[3] Eugene Charniak. *Statistical Language Learning*. MIT Press, Cambridge, MA, 1993.

[4] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Confer-*

ence on Artificial Intelligence, pages 598–603, Menlo Park, 1997. AAAI Press/MIT Press.

[5] Mahesh V. Chitrao and Ralph Grishman. Statistical parsing of messages. In *DARPA Speech and Language Workshop*, pages 263–266, 1990.

[6] Joshua Goodman. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 177–183, 1996.

[7] Joshua Goodman. Global thresholding and multiple-pass parsing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 11–25, 1997.

[8] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[9] Fred Kochman and Joseph Kupin. Calculating the probability of a partial parse of a sentence. In *DARPA Speech and Language Workshop*, pages 273–240, 1991.

[10] David M. Magerman and Mitchell P. Marcus. Parsing the voyager domain using pearl. In *DARPA Speech and Language Workshop*, pages 231–236, 1991.

[11] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19:313–330, 1993.

[12] Scott Miller and Heidi Fox. Automatic grammar acquisition. In *Proceedings of the Human Language Technology Workshop*, pages 268–271, 1994.