Robust Parsing Using a Hidden Markov Model

Wide R. Hogenhout Yuji Matsumoto

Nara Institute of Science and Technology

Abstract. Recent approaches to statistical parsing include those that estimate an approximation of a stochastic, lexicalized grammar directly from a treebank and others that rebuild trees with a number of tree-constructing operators, which are applied in order according to a stochastic model when parsing a sentence. In this paper we take an entirely different approach to statistical parsing, as we propose a method for parsing using a Hidden Markov Model. We describe the stochastic model and the tree construction procedure, and we report results on the Wall Street Journal Corpus.

1 Introduction

Recent approaches to statistical parsing include those that estimate an approximation of a stochastic, lexicalized grammar directly from a treebank (Collins, 1997; Charniak, 1997) and others that rebuild trees with a number of tree-construction operators, which are applied in order according to a stochastic model when parsing a sentence (Magerman, 1995; Ratnaparkhi, 1997). The results have been around 86% in labeled precision and recall on the Wall Street Journal treebank (Marcus, Santorini, and Marcinkiewicz, 1994).
In this paper we take an entirely different approach to statistical parsing. We propose a method for left to right parsing using a Hidden Markov Model (HMM). The results we obtain are not as good as the more general approaches mentioned above, which consider the whole sentence rather then working in an incremental fashion, but the method does give a number of interesting new perspectives. In particular, it can be applied in an environment that requires left to right processing, such as a speech recognition system, it can easily process text that has not been separated into sentences (for example when punctuation is missing or when processing ungrammatical, spoken text), and it can give a shallow parse (i.e., leaving out long distance dependencies) as it is focused on local context. It also makes the parsing process closer to the way humans process language, although we do not explore this psychological aspect in this paper.
In the next three sections we will discuss the way we decide the syntactic context of a word ("traversal strings"), how this can be used for parsing and how a tree can be constructed from them. The following four sections discuss the HMM model used to predict a syntactic context for every word. The last two sections discuss the results, conclusions and future perspectives.

2 Traversal Strings

Take the sentence "I am singing in the rain." This can be analyzed as indicated in figure 1, where the first line of symbols above the text indicates parts of speech as used in the Wall Street Journal Corpus (for example, VBG stands for "verb - gerund or present participle"). The abbreviations used for nonterminals are self-explanatory.
We would like to characterize every word separately instead of having one intertwined structure that models everything. This is possible by tracing the path from every word through the tree up to the top of the tree. This results in table 1.

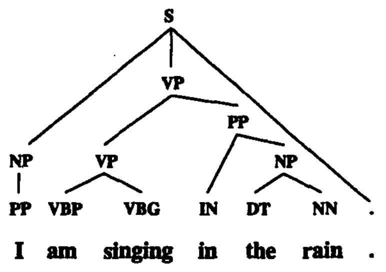

Figure 1. Example of Syntactic Analysis.

We will call these strings (excluding the word and its tag) traversal strings. It will be obvious that this representation is very redundant, and it is exactly this property that we will exploit. But first we note it is also possible to carry out the inverse action of reconstructing the tree from a set of traversal strings. Later we will describe a robust, heuristic algorithm for reconstructing the tree.

The basic concept of parsing with traversal strings is that after seeing a word, one considers a number of possible tree contexts in which that word normally occurs. The most likely ones are selected both by considering the likelihood of a context occurring with a word and the likelihood of a context following another context. As for the last relation, it is here that the redundance becomes meaningful; since neighboring traversal strings are often partially or completely equal.

Oflazer (1996) used a similar structure called "vertex lists" which he defined as the path from a leaf to the root of the tree but, different from our definition, including the tag and the word. Oflazer uses vertex lists for error-tolerant tree-matching. In some cases trees can be said to match approximately, and using vertex lists to quantify the amount of difference between trees, Oflazer shows how trees similar to a given tree can be retrieved from a database (treebank). As will be clear from what follows, we use traversal strings in a completely different way, and to the best of our knowledge these strings have not been used for parsing before.

The work of Joshi and Srinivas (1994) actually comes closer to our work. While we attach traversal strings to word-tag pairs, they do the same with elementary structures in a Lexicalized Tree-Adjoining Grammar, called Supertags. This is a partial tree that only contains one lexical word, as well as part of its context. Joshi and Srinivas show how, by statistically choosing such structures for each word in a sentence, they are able to disambiguate syntactical structure. Note however that the results they give are not comparable to ours as they only tested on short sentences, and report on the accuracy of Supertags rather than bracket-accuracy or recall.

```
I/PP -> NP -> S
am/VBP -> VP -> VP -> S
singing/VBG -> VP -> VP -> S
in/IN -> PP -> VP -> S
the/DT -> NP -> PP -> VP -> S
rain/NN -> NP -> PP -> VP -> S
./. -> S
```

Table 1. Example of Traversal Strings

3 Parsing with Traversal Strings

Figure 2 shows the system design. When presented with a sentence, the first component finds the most likely set of part of speech tags (not shown) and traversal strings matching the words in the sentence. After this a second component assembles a tree from the traversal strings.

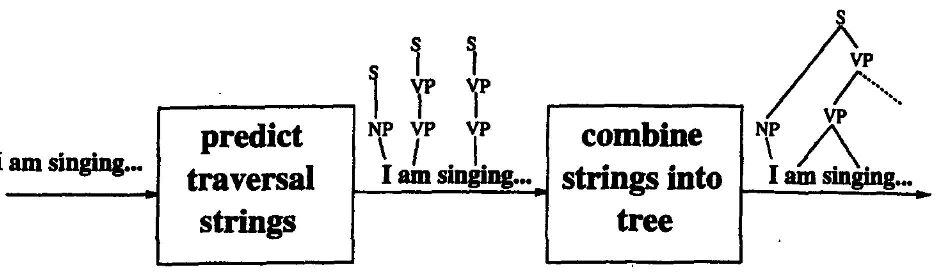

Figure 2. System Design

The prediction of traversal strings presents us with a problem, since traversal strings of arbitrary length are too numerous to be predicted accurately. To see our answer to this problem it is instructive to look at the analysis of a longer sentence, please see figure 3. In particular, notice the shaded areas that show how traversal strings of neighboring words are often equal or partially equal. The most common relation that is seen is a 'shift', where the vertex at position n for word w_i becomes vertex $n+1$ for word w_{i+1}. At the bottom of the traversal string one vertex is added, and nothing else changes.
This illustrates the next step we will take. Even after cutting off traversal strings at a fixed maximum length, it is still possible to reconstruct the tree. The dotted line in figure 3 shows how traversal strings are cut off at a maximum length of 5 vertices. Having part of the traversal string still leaves it possible to see that a particular word is likely to be in the same context as his neighbor. More generally, we look at what subtrees are likely to share part of their context with other, neighboring subtrees. We will show how doing this iteratively makes it possible to restore the tree with a high degree of accuracy.

4 Transforming Traversal Strings into Trees

We use a heuristic algorithm for reconstructing a tree from traversal strings. This includes "partial" traversal strings, but we simply refer to them as traversal strings since we will always be working with partial traversal strings anyway. This is a brief, informal description of the algorithm. A complete technical description is given in (Hogenhout, 1998).
The algorithm is based on the heuristic that best matches should go first. The best match is decided by checking neighboring strings (or, later, subtrees) for equal nonterminals starting at the top of either side. The pair with the most matching nonterminals is merged as displayed in figure 4. This process is repeated until one tree remains, or when there are no more matching neighbors.
For example, the choice

```
in/IN -> PP -> VP -> S
the/DT -> NP -> PP -> VP -> S
rain/NN -> NP -> PP -> VP -> S
```

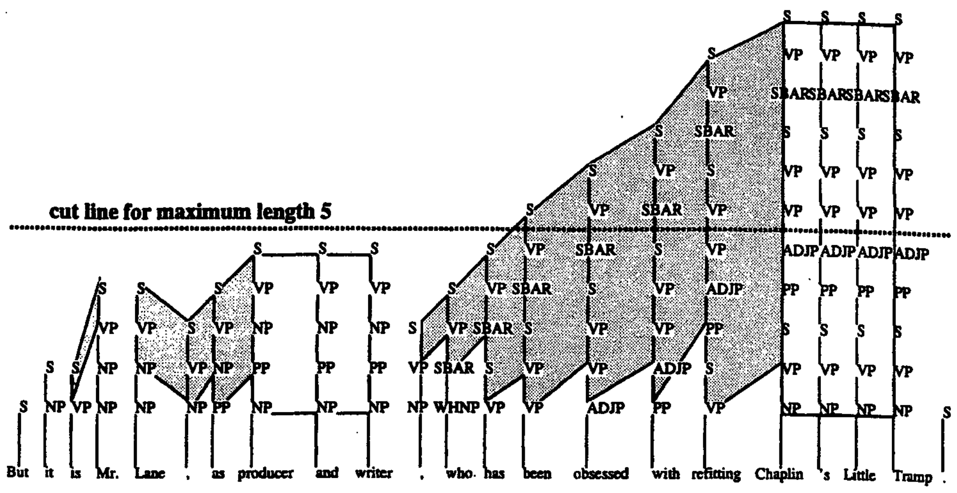
cut line for maximum length 5
But it is Mr. Lane , as producer and writer , who has been obsessed with refitting Chaplin 's Little Tramp .

Figure 3. Similarity between Traversal Strings and cut at maximum length 5

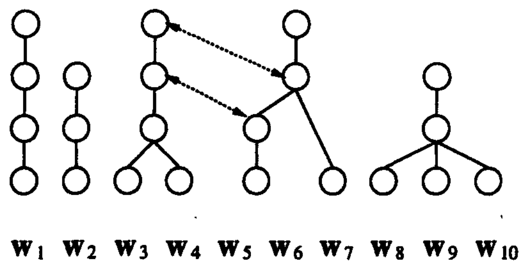
W_1 W_2 W_3 W_4 W_5 W_6 W_7 W_8 W_9 W_{10}

Figure 4. Traversal Strings merged into (sub)trees

would initially be decided in favor of *the* and *rain*, and after they are merged completely, the top three nonterminals would be merged to those of *in*.
There is an easy way of testing this algorithm. One can take trees from a treebank, convert the trees to traversal strings, then use the algorithm to reconstruct the trees. Figure 5 shows the labeled accuracy and recall of these reconstructed trees when compared to the original treebank trees, for various maximum traversal string lengths.
The accuracy is calculated as

$$\text{accuracy} = \frac{\text{number of identical brackets with identical nonterminal}}{\text{number of brackets in system parse}} \qquad (1)$$

and the recall as

$$\text{recall} = \frac{\text{number of identical brackets with identical nonterminal}}{\text{number of brackets in treebank parse}} \qquad (2)$$

which we will refer to as "labeled accuracy" and "labeled recall" as opposed to the "unlabeled" versions of these measures that ignore nonterminals.

Even for long traversal strings the original tree is not reconstructed completely. This happens, for example, when two identical nonterminals are siblings, as in the sentence "*She gave* [NP *the man*] [NP *a book*]." It is of course possible to solve such problems with a post-processor that tries to recognize such situations and correct them whenever they arise. But as can be seen it only involves a small percentage (about 2%) of all brackets and for this reason it is not very significant at this stage.

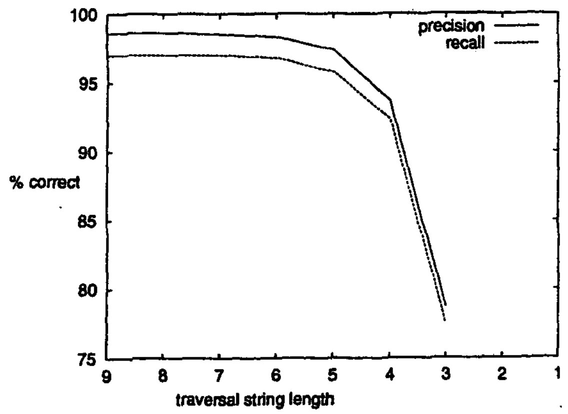

Figure 5. Upper Bound Imposed by Tree Construction Algorithm

The graph shows that if we are capable of predicting up to 5 or more vertices, the algorithm will be able to do very well. If we can only predict up to 4 vertices we still have a high upper bound, but it is slightly lower. Predicting up to 3 or less vertices however will not produce useful results.

It must however be stressed that this is only an upper bound and does not reflect the performance of a useful system in any way. The upper bound only helps to pin down the border line of 4-5 vertices, and what really counts in practice is how the algorithm will do when the traversal strings that are predicted contain errors—as they undoubtedly will.

5 Guessing Traversal Strings

We will now look at the question of how to predict traversal strings. As will become clear when inspecting the equation this bears similarity to part-of-speech tagging. But there is one factor that makes a big difference: we do not test on the correct traversal string, but on the result of the tree that is reconstructed at the end. In many cases the traversal string that is guessed is not correct, but similar to the correct traversal string, and a similar traversal string will render much better results at tree reconstruction than a completely different one.

As usual our approach is maximizing the likelihood of the training data. We will use a Hidden Markov Model which has traversal string-tag combinations as states and which produces words as output. We do not reestimate probabilities using the Baum-Welch algorithm (Baum, 1972) but we use smoothed Maximum Likelihood estimates from treebank data.

Let us say we have a string of words $w_1...w_n$, and we are interested in guessing tags $\mathcal{T} = t_1...t_n$ and traversal strings $\mathcal{S} = s_1...s_n$. We also use $s_0 = t_0 = w_0 = s_{n+1} = t_{n+1} =$ dummy as a short-hand to signal the beginning and end of the sentence.

We take the probability of a sentence to be

$$p(w_1...w_n) = \sum_{\mathcal{T},\mathcal{S}} p(w_1...w_n|\mathcal{T},\mathcal{S}) \tag{3}$$

$$\approx \sum_{\mathcal{T},\mathcal{S}} \prod_{i=0}^{n} p(w_i|s_i,t_i)p(s_{i+1},t_{i+1}|s_i,t_i), \tag{4}$$

corresponding to the transition and output probabilities of a hidden markov model.
In practice the probabilities $p(w_i|s_i,t_i)$ and $p(s_{i+1},t_{i+1}|s_i,t_i)$ can not be estimated directly using Maximum Likelihood because of sparse data. For this reason we smooth the estimates with our version of lower-order models as follows:

$$\tilde{p}(w_i|s_i,t_i) = \lambda_{siti}p(w_i|s_i,t_i) + (1-\lambda_{siti})p(w_i|t_i) \tag{5}$$

where the interpolation factor λ_{siti} is adjusted for different values of s_i and t_i as suggested in (Bahl, Jelinek, and Mercer, 1983). We also divide the s_i-t_i pair values over different *buckets* so that all pairs in the same bucket have the same λ parameter. It should be noted that we have a special word which stands for "unknown word," to take care of words that were not seen in the training data.
We do something similar for $p(s_{i+1},t_{i+1}|s_i,t_i)$, namely

$$\tilde{p}(s_{i+1}t_{i+1}|s_i,t_i) = \delta^1_{siti}p(s_{i+1}t_{i+1}|s_i,t_i) + \delta^2_{siti}p(s_{i+1}t_{i+1}|t_i) + \delta^3_{siti}p(s_{i+1}t_{i+1}) \tag{6}$$

where of course $\delta^1_{siti} + \delta^2_{siti} + \delta^3_{siti} = 1$. The interpolation factors are bucketed in the same way.
Using the obtained model we choose $\mathcal{T}$ and $\mathcal{S}$ by maximizing the probability of the sentence that we wish to analyze:

$$(\mathcal{T},\mathcal{S})^* = \operatorname*{argmax}_{(\mathcal{T},\mathcal{S})} p((\mathcal{T},\mathcal{S})|w_1...w_n) \tag{7}$$

$$\approx \operatorname*{argmax}_{(\mathcal{T},\mathcal{S})} \prod_{i=0}^{n} p(w_i|s_i,t_i)p(s_{i+1},t_{i+1}|s_i,t_i) \tag{8}$$

which can be resolved using the Viterbi-algorithm (Viterbi, 1967).

6 Selection of Part of Speech Tags

The process outlined above still has one problem that will be central in the rest of the discussion. The number of traversal strings is easily a few thousand, and the number of part of speech tag-traversal string pairs is even larger. Clearly, the computational complexity of the algorithm is in calculating (8). But, given a word and the history up to that word, most tags and traversal strings can be ruled out immediately. We will therefore only consider a fraction of the possible part of speech tags and traversal strings. This section will discuss how we select part of speech tags.
The equation we use for selecting a tag is similar to the standard tagging HMM based model. We pretend for the time being that we are dealing with another stochastic process, namely one that only generates tags. We assume that

$$p(w_1...w_n) = \sum_{\mathcal{T}} p(w_1...w_n,\mathcal{T}) \tag{9}$$

$$\approx \sum_{\mathcal{T}} \prod_{i=0}^{n} p(w_i|t_i)p(t_{i+1}|t_i) \tag{10}$$

but we do not really use this model, we only use the idea behind it to approximate the probability of a tag. We find the most likely tags after seeing word i using the following approximation:

$$
\begin{aligned}
t_i^* &= \operatorname*{argmax}_t p(t_i = t | w_1 ... w_i) && (11)\\
&\approx \operatorname*{argmax}_t \sum_{u_{i-1}} p(t_i = t | t_{i-1} = u_{i-1}) p(w_i | t_i) \alpha_t (i-1, u_{i-1}) && (12)\\
&\approx \operatorname*{argmax}_t \sum_{u_{i-1}} p(t_i = t | t_{i-1} = u_{i-1}) p(w_i | t_i)\\
&\qquad \sum_{(s,u)\in B_{i-1}} \alpha_{(s,u)}(i-1, (s,u)) \delta(u, u_{i-1}) && (13)
\end{aligned}
$$

where s is a traversal string, the symbol B_{i-1} indicates the set of tag-traversal string pairs that is being considered for word w_{i-1}, and α indicates the "forward probability" according to the HMM. As usual $\delta(u, u_{i-1}) = 1$ if $u = u_{i-1}$ and 0 otherwise. We will discuss later how the set B_{i-1} is chosen, but this of course depends on the tags selected for the word w_{i-1}. We distinguish between α_t (tagging model) and $\alpha_{(s,u)}$ (traversal string model).
We take two significant assumptions at this point. First, we do not really use the HMM indicated in (10), but in equation (12) we restrict ourselves to the forward probability. The second assumption we take is (13), i.e., we estimate the probability of the previous tag by the tag-traversal string pairs that were selected for the previous word. Using this method we do not need to implement the markov model for tags, we only need the tables for $p(t_i|t_{i-1})$ and $p(w_i|t_i)$. As we already need the last one for the traversal string model, we only need the (small) table $p(t_i|t_{i-1})$ especially for tagging.
We must emphasize that the tagging described here is only a first estimate. We consider the most likely one, two or three tags according to this model and discard the rest. Once they are selected, these probabilities are discarded and we return to the regular model. The next section will describe how the tags are selected in the next phase.

7 Selection of Traversal Strings : First Phase

The next problem is how to select a few traversal strings given a word and a few tags, one of which is likely to be correct. The model we use for this pre-selection is actually more simple; as we ignore the selected traversal strings for previous words.[1] From the corpus we directly estimate in Maximum Likelihood fashion

$$P(w_i, s_i, t_i) \tag{14}$$

and select the most likely traversal strings s_i from this table. If there are too few samples for a particular word w_i, the list is completed with the more general distribution

$$P(s_i, t_i), \tag{15}$$

again maximizing over s_i. We will have to consider that we do not have a single tag but several options, but we will first pretend that we do have one single tag.
Figure 6 shows the results of this first phase, in case the maximum length of traversal strings is set to 5. If the best 50 candidates are selected according to (14), supplemented with selection according to (15) if necessary, we have the correct candidate between them about 80% of the time. That means that for 20% of the words, we can only hope that a similar traversal string will be available for them. If we use the best 300 candidates, we will miss the correct candidate for about one word per sentence. We must however emphasize two points:

1. The question is not only if we can select the correct candidate. It is crucial that, when a wrong candidate is chosen, this is at least similar to the correct candidate.
2. Figure 6 indicates the percentage for traversal strings cut of at length 5. If traversal stings of a different maximum length are used, this will change (the higher the maximum length, the lower the percentage of hits).

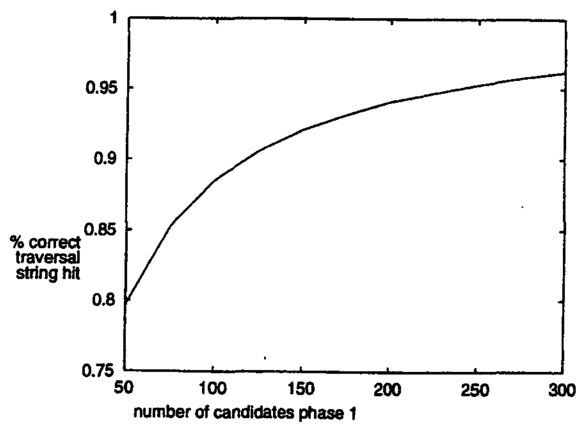

Figure 6. Hit percentage for first phase

Now we return to the tagging problem; after all we do not have the right tag available to us. We solve this, heuristically, as follows. Let a be the most likely tag, b the second most likely and c the third.

- If $p(a)/p(b) > 50$, select 300 candidates for tag a and ignore other tags.
- If $50 \geq p(a)/p(b) > 4$ we select 300 candidates for tag a and more 100 candidates for tag b.
- If $4 \geq p(a)/p(b)$ we select 300 candidates for tag a, 200 candidates for tag b and 100 candidates for tag c.

This scheme gives more candidates for more ambiguous words, but as about 80% of all words fall in the first category and only 9% in the last category, this is not so bad. This list will contain the correct traversal string about 95% of the time.

8 Selection of Traversal Strings : Second Phase

The previous section explained how initial candidates can be selected quickly from all possible sets. After these initial candidates were selected, the transition and output probabilities are calculated. Let again B_{i-1} be the set of candidates considered for word w_{i-1}. Then we need to calculate (regrouping the product as compared to (8)) the quantity

$$\alpha(s_i t_i) = \sum_{(s,t)\in B_{i-1}} \alpha(s,t)p(w_i|s_i,t_i)p(s_i,t_i|s,t) \tag{16}$$

where we set

$$\alpha(s_0 t_0) = \begin{cases} 1 \text{ if } s_0 = \text{dummy and } t_0 = \text{dummy} \\ 0 \text{ otherwise} \end{cases} \tag{17}$$

and $B_0 = \{(\text{dummy},\text{dummy})\}$. The sum in (16) reflects almost all of the time that the calculation process takes up. But equation (16) gives a much more accurate estimate of likelihood than the rather primitive word-based selection (14), so once this sum is calculated we have a much better idea of the likelihood of candidates. For this reason we use two criteria:

[1] Note that using a technique similar to that for part of speech tags is not an option as this is exactly what we are trying to avoid doing for all possible traversal strings.

- In the first phase we use equation (14) and select the best μ candidates. (As explained, depending on tagging confidence we vary the number of candidates, so μ should be thought of as an average.)
- In the second phase we use equation (16) and select the best η candidates.

It will be clear that we can choose $\eta \ll \mu$. We have illustrated this in figure 7, which displays the percentage of correct candidates for various values for η, again using a maximum traversal string length of 5. Note that the computational complexity of the Viterbi algorithm will be $O(\mu\eta n)$ where n is sentence length.

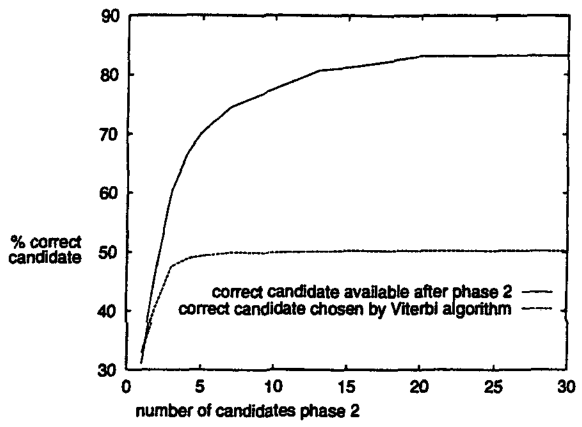

Figure 7. Hit percentage for second phase

Figure 7 gives the percentage of cases in which the correct candidate is available (the upper line) and also the percentage of cases in which the correct candidate is chosen by the Viterbi algorithm. A remarkable fact arises from this figure: the percentage of traversal strings that are chosen correctly stabilizes at about $\eta = 4$. From that point the percentage is about 50% and while increasing η increases the chance that the correct one will be available, choosing it becomes more difficult and these two effects cancel each other out. Nevertheless the result continues to improve for higher η, as better alternatives become available. We will put η to 15 as a higher number contributes little more to the final scores.

9 Parsing

We have now dealt with all parts of the parsing process. Whenever a new word is seen, a few tags are selected according to (13). After this a set of about 300 (depending on the confidence in the tags) traversal strings is selected according to (14) or (15). The forward probability of these candidates is calculated (16) and this is used to further reduce the candidates to 15 tag-traversal string pairs. This set is saved with their forward probabilities, and when the end-of-sentence signal is received the best series is given by the Viterbi algorithm. A tree is then produced according to the algorithm described in section 4.
Until now we have set the maximum traversal string length to 5 but now we can show how variation in the maximum length affects the result. The experiments we present here were carried out with data from the Wall Street Journal Treebank. Parameters were estimated with the first 22 sections (over 40,000 sentences), section 24 was used for smoothing (interpolation)

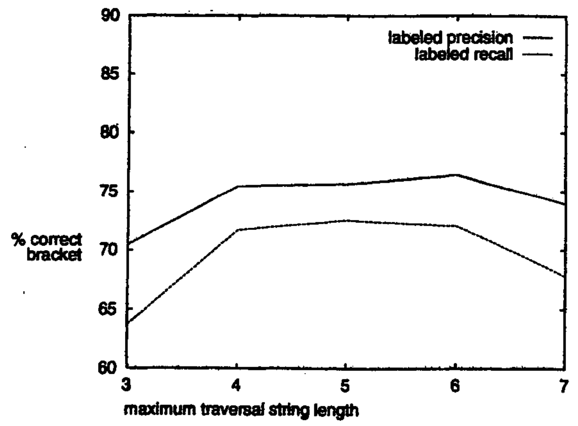

Figure 8. Precision and recall plotted against maximum traversal string length

and section 23 (2,416 sentences) was used exclusively for testing. Figure 8 shows the labeled accuracy and recall for various maximum lengths that result from this data.
This shows that the optimal length is about 4, 5 or perhaps 6. This picture is slightly favoring the shorter lengths, since μ and η are fixed while the longer lengths have more candidates to choose from. But on the other hand, keeping μ and η fixed corresponds to giving the algorithm a certain time and letting it do its best in the given time. The longer lengths also have a disadvantage in that they lead to larger tables, using more memory.
The differences between 4, 5 and 6 are minor, and the performance degrades seriously at 3 or 7. This shows that a maximum of 5 is a sensible choice. The first column of table 2 gives detailed information about the final performance. It is also possible to restrict the parser to lower level structures, taking only those parts which are the most safe, namely low-level brackets that do not depend on long distance dependencies. We carried this out by removing brackets covering more than three words and some particular nonterminals that often result in errors, such as SBAR. These results are indicated in the "Shallow Parsing" column.

Table 2. Parsing Results

Measure	Regular Score	Shallow Parsing
labeled precision	75.6%	87.4%
labeled recall	72.9%	37.9%
unlabeled precision	79.5%	89.2%
unlabeled recall	76.6%	38.9%
crossing brackets per sentence	2.31	0.44
tagging accuracy	94.4%	
speed on Sparc Station 20	6.5 words/second	

10 Discussion

The method we propose analyses language indirectly as a regular language. This makes it impossible to use long distance dependencies, but nevertheless the experiment shows that it

performs quite reasonable and is very robust.
The score is less than the scores obtained by systems that consider the entire sentence with, in particular, the headwords of phrases. But the method creates new possibilities such as processing ungrammatical text and processing unpunctuated text. Shallow parsing is also a possible application.
As far as future directions are concerned, we would like to mention that our parsing strategy is not limited to regular languages and HMM models. It is possible to switch to a history-based approach, where the choice of s_i depends on both the words $w_1 ... w_i$ and all earlier tags and traversal strings chosen by the system. In that case a statistical decision tree or a markov field can be used to model the optimal choice for s_i after seeing word w_i.

11 Acknowledgements

We would like to thank the anonymous reviewers for their useful comments.

References

Bahl, Lalit R., Frederick Jelinek, and Robert L. Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):179–190.
Baum, L. E. 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8.
Charniak, E. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI*, pages 598–603.
Collins, M. J. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23.
Hogenhout, Wide R. 1998. *Supervised Learning of Syntactic Structure.* Ph.D. thesis, Nara Institute of Science and Technology.
Joshi, Aravind K. and B. Srinivas. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*, pages 154–160.
Magerman, D. M. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33d Annual Meeting of the Association for Computational Linguistics*, pages 276–283.
Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.
Oflazer, Kemal. 1996. Error-tolerant tree matching. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 860–864.
Ratnaparkhi, Adwait. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing.*
Viterbi, A. J. 1967. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269.