Lookahead in Deterministic Left-Corner Parsing[*]

James HENDERSON
School of Informatics, University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW
United Kingdom
james.henderson@ed.ac.uk

Abstract

To support incremental interpretation, any model of human sentence processing must not only process the sentence incrementally, it must to some degree restrict the number of analyses which it produces for any sentence prefix. Deterministic parsing takes the extreme position that there can only be one analysis for any sentence prefix. Experiments with an incremental statistical parser show that performance is severely degraded when the search for the most probable parse is pruned to only the most probable analysis after each prefix. One method which has been extensively used to address the difficulty of deterministic parsing is lookahead, where information about a bounded number of subsequent words is used to decide which analyses to pursue. We simulate the effects of lookahead by summing probabilities over possible parses for the lookahead words and using this sum to choose which parse to pursue. We find that a large improvement is achieved with one word lookahead, but that more lookahead results in relatively small additional improvements. This suggests that one word lookahead is sufficient, but that other modifications to our left-corner parsing model could make deterministic parsing more effective.

1 Introduction

Incremental interpretation is a fundamental property of the human parsing mechanism. To support incremental interpretation, any model of sentence processing must not only process the sentence incrementally, it must to some degree restrict the number of analyses which it produces for any sentence prefix. Otherwise the ambiguity of natural language would make the number of possible interpretations at any point in the parse completely overwhelming. Deterministic parsing takes the extreme position that there can only be one analysis for any sentence prefix. We investigate methods which make such a strong constraint feasible, in particular the use of lookahead.

In this paper we do not try to construct a single deterministic parser, but instead consider a family of deterministic parsers and empirically measure the optimal performance of a deterministic parser in this family. As has been previously proposed by Brants and Crocker (2000), we take a corpus-based approach to this empirical investigation, using a previously defined statistical parser (Henderson, 2003). The statistical parser uses an incremental history-based probability model based on left-corner parsing, and the parameters of this model are estimated using a neural network. Performance of this basic model is state-of-the-art, making these results likely to generalize beyond this specific system.

We specify the family of deterministic parsers in terms of pruning the search for the most probable parse. Both deterministic parsing and the use of k-word lookahead are characterized as constraints on pruning this search. We then derive the optimal pruning strategy given these constraints and the probabilities provided by the statistical parser's left-corner probability model. Empirical experiments on the accuracy of a parser which uses this pruning method indicate the best accuracy we could expect from a deterministic parser of this kind. This allows us to compare different deterministic parsing methods, in particular the use of different amounts of lookahead.

In the remainder of this paper, we first discuss how the principles of deterministic parsing can be expressed in terms of constraints on the search strategy used by a statistical parser. We then present the probability model used by the statistical parser, the way a neural network is used to estimate the parameters of this proba-

[*] This work has been supported by the Department of Computer Science, University of Geneva.

bility model, and the methods used to search for the most probable parse according these parameters. Finally, we present the empirical experiments on deterministic parsing with lookahead, and discuss the implications of these results.

2 Approximating Optimal Deterministic Parsing

The general principles of deterministic parsing, as proposed by Marcus (1980), are that parsing proceeds incrementally from left to right, and that once a parsing decision has been made, it cannot be revoked or overridden by an alternative analysis. We translate the first principle into the design of a statistical parser by using an incremental generative probability model. Such a model provides us with probabilities for partial parses which generate prefixes of the sentence and which do not depend on the words not in this prefix. We can then translate the second principle into constraints on how a statistical parser chooses which partial parses to pursue further as it searches for the most probable complete parse.

The principle that decisions cannot be revoked or overridden means that, given a sequence of parser actions $a_1,...,a_{i-1}$ which we have already chosen, we need to choose a single parser action a_i before considering any subsequent parser action a_{i+1}. However, this constraint does not prevent considering the effects of multiple alternative parser actions for a_i before choosing between them. This leaves a great deal of flexibility for the design of a deterministic parser, because the set of actions defined by a deterministic parser does not have to be the same as the basic decisions defined by our probability model. We can combine any finite sequence of decisions $d_j,...,d_{j+l}$ from our probability model into a single parser action a_i. This combination allows a deterministic parser to consider the effects of the entire sequence of decisions $d_j,...,d_{j+l}$ before deciding whether to choose it. Different deterministic parser designs will combine the basic decisions into parser actions in different ways, thereby imposing different constraints on how long a sequence of future decisions $d_j,...,d_{j+l}$ can be considered before choosing a parser action.

Once we have made a distinction between the basic decisions of the probability model d_j and the parser actions $a_i = d_j,...,d_{j+l}$, it is convenient to express the choice of the parse $a_1,...,a_n$ as a search for the most probable $d_1,...,d_m$, where $a_1,...,a_n = d_1,...,d_m$. The search incrementally constructs partial parses and prunes this search down to a single partial parse after each complete parser action. In other words, given that the search has so far chosen the partial parse $a_1,...,a_{i-1} = d_1,...,d_{j-1}$, the search first considers all the possible partial parses $d_1,...,d_{j-1},d_j,...,d_{j+l}$ where there exists an $a_i = d_j,...,d_{j+l}$. The search is then pruned down to only the best $d_1,...,d_{j-1},d_j,...,d_{j+l}$ from this set, and the search continues with all partial parses containing this prefix. Thus the search is allowed to delay pruning for as many basic decisions as are combined into a single parser action.

Rather than considering one single deterministic parser design, in this paper we consider a family of deterministic parser designs. We then determine tight upper bounds on the performance of any deterministic parser in this family. We define a family of deterministic parsers by starting with a particular incremental generative probability model, and consider a range of ways to define parser action a_i as finite sequences $d_j,...,d_{j+l}$ of these basic decisions.

We define the family of parser designs as allowing the combination of any sequence of decisions which occur between the parsing of two words. After a word has been incorporated into the parse, this constraint allows the search to consider all the possible decision sequences leading up to the incorporation of the next word, but not beyond. When the next word is reached, the search must again be pruned down to a single analysis. This is a natural point to prune, because it is the position where new information about the sentence is available. Given this definition of the family of deterministic parsers and the fact that we are only concerned with an upper bound on a deterministic parser's performance, there is no need to consider parser designs which require more pruning than this, since they will never perform as well as a parser which requires less pruning.

Unfortunately, allowing the combination of any sequence of decisions which occur between the parsing of two words does not exactly correspond to the constraints on deterministic parsing. This is because we cannot put a finite upper bound on the number of actions which occur between two words. Thus this class of parsers includes non-deterministic parsers, and therefore our performance results represent only an upper bound on the performance which could be achieved by a deterministic parser in the class.

However, there is good reason to believe this is a tight upper bound. Lexicalized theories of syntax all assume that the amount of information about the syntactic structure contributed by each word is finite, and that all the information in the syntactic structure is contributed by some word. Thus it should possible to distribute all the information about the structure across the parse in such a way that a finite amount falls in between each word. The parsing order we use (a form of left-corner parsing) seems to achieve this fairly well, except for the fact that it uses a stack. Parsing right-branching structures, such as are found in English, results in the stack growing arbitrarily large, and then the whole stack needs to be popped at the end of the sentence. With the exception of these sequences of popping actions, the number of actions which occur between any two words could be bounded. In our training set, the bound on the number of non-popping actions between any two words could be set at just 4.

In addition to designing parser actions to make deterministic parsing easier, another mechanism which is commonly used in deterministic parser designs is lookahead. With lookahead, information about words which have not yet been incorporated into the parse can be used to decide what action to choose next. We consider models where the lookahead consists of some small fixed-length prefix of the un-parsed portion of the sentence, which we call k-word lookahead. This mechanisms is constrained by the requirement that the parser be incremental, since a deterministic parser with k-word lookahead can only provide an interpretation for the portion of the sentence which is k words behind what has been input so far. Thus it is not possible to include the entire unboundedly-long sentence in the lookahead. The family of deterministic parsers with k-word lookahead would include parsers which sometimes choose parser actions without waiting to see all k words (and thus on average allow interpretation sooner), but because here we are only concerned with the optimal performance achievable with a given lookahead, we do not have to consider these alternatives.

The optimal deterministic parser with lookahead will choose the partial parse which is the most likely to lead to the correct complete parse given the previous partial parse plus the k words of lookahead. In other words, we are trying to maximize $P(a_{t+1}|a_1,...,a_t,w_{t+1},...,w_{t+k})$, which is the same as maximizing $P(a_{t+1},w_{t+1},...,w_{t+k}|a_1,...,a_t)$ for the given $w_{t+1},...,w_{t+k}$. (Note that any partial parse $a_1,...,a_t$ generates the words $w_1,...,w_t$, because the optimal deterministic parser designs we are considering all have parser actions which combine the entire portion of a parse between one word and another.) We can compute this probability by summing over all parses which include the partial parse $a_1,...,a_{t+1}$ and which generate the lookahead string $w_{t+1},...,w_{t+k}$.

$$P(a_{t+1},w_{t+1},...,w_{t+k}|a_1,...,a_t) = \sum_{(a_{t+2},...,a_{t+k})} P(a_{t+1},a_{t+2},...,a_{t+k}|a_1,...,a_t)$$
$$\text{where } a_{t+1},...,a_{t+k} \text{ generates } w_{t+1},...,w_{t+k}\ .$$

Because the parser actions are defined in terms of basic decisions in the probability model, we can compute this sum directly using the probability model. A real deterministic parser cannot actually perform this computation explicitly, because it involves pursuing multiple analyses which are then discarded. But ideally a deterministic parser should compute an estimate which approximates this sum. Thus we can compute the performance of a deterministic parser which makes the ideal use of lookahead by explicitly computing this sum. Again, this will be an upper bound on the performance of a real deterministic parser, but we can reasonably expect that a real deterministic parser can reach performance quite close to this ideal for a small amount of lookahead.

This approach to lookahead can also be expressed in terms of pruning the search for the best parse. After pruning to a single partial parse $a_1,...,a_t$ which ends by generating w_t, the search is allowed to pursue multiple parses in parallel until they generate the word w_{t+k}. The probabilities for these new partial parses are then summed to get estimates of $P(a_{t+1},w_{t+1},...,w_{t+k}|a_1,...,a_t)$ for each possible a_{t+1}, and these sums are used to choose a single a_{t+1}. The search is then pruned by removing all partial parses which do not start with $a_1,...,a_{t+1}$. The remaining partial parses are then continued until they generate the word w_{t+k+1}, and their probabilities are summed to decide how to prune to a single choice of a_{t+2}.

By expressing the family of deterministic parsers with lookahead in terms of a pruning strategy on a basic parsing model, we are able to easily investigate the effects of different lookahead lengths on the maximum performance of a deterministic parser in this family. To complete the specification of the family of deterministic parsers, we simple have to specify the basic parsing model, as done in the next section.

3 A Generative Left-Corner Probability Model

As with several previous statistical parsers (Collins, 1999; Charniak, 2000), we use a generative history-based probability model of parsing. Designing a history-based model of parsing involves two steps, first choosing a mapping from the set of phrase structure trees to the set of parses, and then choosing a probability model in which the probability of each parser decision is conditioned on the history of previous decisions in the parse. For the model to be generative, these decisions must include predicting the words of the sentence. To support incremental parsing, we want to map phrase structure trees to parses which predict the words of the sentence in their left-to-right order. To support deterministic parsing, we want our parses to specify information about the phrase structure tree at appropriate points in the sentence. For these reasons, we choose a form of left-corner parsing (Rosenkrantz and Lewis, 1970).

In a left-corner parse, each node is introduced after the subtree rooted at the node's first child has been fully parsed. Then the subtrees for the node's remaining children are parsed in their left-to-right order. In the form of left-corner parsing we use, parsing a constituent starts by pushing the leftmost word w of the constituent onto the stack with a *shift(w)* action. Parsing a constituent ends by either introducing the constituent's parent nonterminal (labeled Y) with a *project(Y)* action, or attaching to the parent with a *attach* action.

More precisely, this parsing strategy is a version of left-corner parsing which first applies right-binarization to the grammar, as is done in (Manning and Carpenter, 1997) except that we binarize down to nullary rules rather than to binary rules. This means that choosing the children for a node is done one child at a time, and that ending the sequence of children is a separate choice. We also extended the parsing strategy slightly to handle Chomsky adjunction structures (i.e. structures of the form [X [X ...] [Y ...]]) as a special case. The Chomsky adjunction is removed and replaced with a special "modifier" link in the tree (becoming [X ... [$_{mod}$ Y ...]]). This means that the parser's set of basic actions includes *modify*, as well as *attach*, *shift(w)*, and *project(Y)*. We also compiled some frequent chains of non-branching nodes (such as [S [VP ...]]) into a single node with a new label (becoming [S-VP ...]). All these grammar transforms are undone before any evaluation of the output trees is performed.

Because this mapping from phrase structure trees to sequences of parser decisions is one-to-one, finding the most probable phrase structure tree is equivalent to finding the parse $d_1,..., d_m$ which maximizes $P(d_1,..., d_m)$, as is done in generative models. Because this probability includes the probabilities of the *shift(w_i)* decisions, this is the joint probability of the phrase structure tree and the sentence. The probability model is then defined by using the chain rule for conditional probabilities to derive the probability of a parse as the multiplication of the probabilities of each decision d_i conditioned on that decision's prior parse history $d_1,..., d_{i-1}$.

$$P(d_1,..., d_m) = \Pi_i P(d_i|d_1,..., d_{i-1})$$

The parameters of this probability model are the $P(d_i|d_1,..., d_{i-1})$. Generative models are the standard way to transform a parsing strategy into a probability model, but note that we are not assuming any bound on the amount of information from the parse history which might be relevant to each parameter.

4 Estimating the Parameters with a Neural Network

The most challenging problem in estimating $P(d_i|d_1,..., d_{i-1})$ is that the conditional includes an unbounded amount of information. The parse history $d_1,..., d_{i-1}$ grows with the length of the sentence. In order to apply standard probability estimation methods, we use neural networks to induce finite representations of this sequence, which we will denote $h(d_1,..., d_{i-1})$. The neural network training methods we use try to find representations which preserve all the information about the sequences which are relevant to estimating the desired probabilities.

$$P(d_i|d_1,..., d_{i-1}) \approx P(d_i|h(d_1,..., d_{i-1}))$$

Of the previous work on using neural networks for parsing natural language, by far the most empirically successful has been the work using Simple Synchrony Networks (Henderson, 2003). Like other recurrent network architectures, SSNs compute a representation of an unbounded sequence by incrementally computing a representation of each prefix of the sequence. At each position i, representations from earlier in the sequence are combined with features of the new position i to produce a vector of real valued features which represent the prefix ending at i. This representation is called a hidden representation. It is analogous to the hidden state of a Hidden Markov Model. As long as the hidden representation for position $i-1$ is always used to compute the hidden representation for position i, any information about the entire sequence could be passed from hidden representation to hidden representation and be included in the hidden representation of that sequence. When these representations are then used to estimate probabilities, this property means that we are not making any a priori hard independence assumptions.

The difference between SSNs and most other recurrent neural network architectures is that SSNs are specifically designed for processing structures. When computing the history representation $h(d_1,..., d_{i-1})$, the SSN uses not only the previous history representation $h(d_1,..., d_{i-2})$, but also uses history representations for earlier positions which are particularly relevant to choosing the next parser decision d_i. This relevance is determined by first assigning each position to a node in the parse tree, namely the node which is on the top of the parser's stack when that decision is made. Then the relevant earlier positions are chosen based on the structural locality of the current decision's node to the earlier decisions' nodes. In this way, the number of representations which information needs to pass through in order to flow from history representation i to history representation j is determined by the structural distance between i's node and j's node, and not just the distance between i and j in the parse sequence. This provides the neural network with a linguistically appropriate inductive bias when it learns the history representations, as explained in more detail in (Henderson, 2003). The fact that this bias is both structurally defined and linguistically appropriate is the reason that this parser performs so much better than previous attempts at using neural networks for parsing, such as (Costa et al., 2001).

Once it has computed $h(d_1,..., d_{i-1})$, the SSN uses standard methods (Bishop, 1995) to estimate a probability distribution over the set of possible next decisions d_i given these representations. This involves further decomposing the distribution over all possible next parser actions into a small hierarchy of conditional probabilities, and then using log-linear models to estimate each of these conditional probability distributions. The input features for these log-linear models are the real-valued vectors computed by $h(d_1,..., d_{i-1})$, as explained in more detail in (Henderson, 2003).

As with many other machine learning methods, training a Simple Synchrony Network involves first defining an appropriate learning criteria and then performing some form of gradient descent learning to search for the optimum values of the network's parameters according to this criteria. We use the on-line version of Backpropagation to perform the gradient descent. This learning simultaneously tries to optimize the parameters of the output computation and the parameters of the mapping $h(d_1,..., d_{i-1})$. With multi-layered networks such as SSNs, this training is not guaranteed to converge to a global optimum, but in practice a network whose criteria value is close to the optimum can be found.

5 Searching for the most probable parse

As discussed in section 2, we investigate deterministic parsing by translating the principles of deterministic parsing into properties of the pruning strategy used to search the space of possible parses. The complete parsing system alternates between using the search strategy to decide what partial parse $d_1,..., d_{i-1}$ to pursue further and using the SSN to estimate a probability distribution $P(d_i|d_1,..., d_{i-1})$ over possible next decisions d_i. The probabilities $P(d_1,..., d_i)$ for the new partial parses are then just $P(d_1,..., d_{i-1}) \times P(d_i|d_1,..., d_{i-1})$. When no pruning applies, the partial parse with the highest probability is chosen as the next one to be extended.

Even in the non-deterministic version of the parser, we need to prune the search space. This is because the number of possible parses is exponential in the length of the sentence, and we cannot use dynamic programming to compute the best parse efficiently because we do not make any independence assumptions. However, we have found that the search can be drastically pruned without loss in accuracy, using a similar approach to that used here to model deterministic parsing. After the prediction of each word, we prune all partial parses except a fixed beam of the most probable partial parses. Due to the use of the above left-corner parsing order, we have found that the beam can be as little as 100 parses without having any measurable effect on accuracy. Below we will refer to this beam width as the post-word search beam width.

In addition to pruning after the prediction of each word, we also prune the search space in between two words by limiting its branching factor to at most 5. This, in effect, just limits the number of labels considered for each new nonterminal. We found that increasing the branching factor had no effect on accuracy and little effect on speed.

For the simulations of deterministic parsers, we always applied both the above pruning strategies, in addition to the deterministic pruning. This non-deterministic pruning reduces the number of partial parses $a_1,..., a_{t+1},..., a_{t+k}$ whose probabilities are included in the sum used to choose a_{t+1} for the deterministic pruning. This approximation is not likely to have any significant effect on the choice of a_{t+1}, because the probabilities of the partial parses which are pruned by the non-deterministic pruning tend to be very small compared to the most probable alternatives. The non-deterministic pruning also reduces the set of partial parses which are chosen between during the subsequent deterministic pruning. But this undoubtedly has no significant effect, since experimental results have shown that the level of non-deterministic pruning discussed above does not effect performance even without deterministic pruning.

6 The Experiments

To investigate the effects of lookahead on our family of deterministic parsers, we ran empirical experiments on the standard the Penn Treebank (Marcus et al., 1993) datasets. The input to the network is a sequence of tag-word pairs.[1] We report results for a vocabulary size of 508 tag-word pairs (a frequency threshold of 200).

We first trained a network to estimate the parameters of the basic probability model. We determined appropriate training parameters and network size based on intermediate validation results and our previous experience.[2] We trained several networks and chose the best ones based on their validation performance. The best post-word search beam width for the non-deterministic parser was determined on the validation set, which was 100.

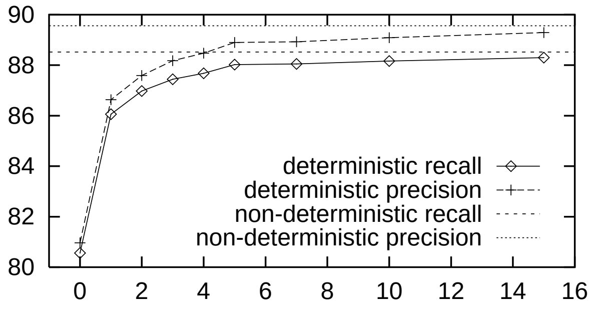

Figure 1: Labeled constituent recall and precision as a function of the number of words of lookahead used by a deterministic parser. Curves reach their non-deterministic performance with large lookahead.

To avoid repeated testing on the standard testing set, we measured the performance of the different models on section 0 of the Penn Treebank (which is not included in either the training or validation sets). Standard measures of accuracy for different lookahead lengths are plotted in figure 1.[3] First we should note that the non-deterministic parser has state-of-the-art accuracy (89.0% F-measure), considering its vocabulary size. A moderately larger vocabulary version (4215 tag-word pairs) of this parser achieves 89.8% F-measure on section 0, where the best current result on the testing set is 90.7% (Bod, 2003).

As expected, the deterministic parsers do worse than the non-deterministic one, and this difference becomes less as the lookahead is lengthened. What is surprising about the curves in figure 1 is that there is a very large increase in performance from zero words of lookahead

[1]We used a publicly available tagger (Ratnaparkhi, 1996) to provide the tags. This tagger is run before the parser, so there may be some information about future words which is available in the disambiguated tag which is not available in the word itself. We don't think this has had a significant impact on the results reported here, but currently we are working on doing the tagging internally to the parser to avoid this problem.

[2]The best network had 80 hidden units for the history representation. Weight decay regularization was applied at the beginning of training but reduced to near 0 by the end of training. Training was stopped when maximum performance was reached on the validation set, using a post-word beam width of 5.

[3]All our results are computed with the evalb program following the standard criteria in (Collins, 1999). We used the standard training (sections 2–22, 39,832 sentences, 910,196 words) and validation (section 24, 1346 sentence, 31507 words) sets (Collins, 1999). Results of the nondeterministic parser average 0.2% worse on the standard testing set, and average 0.8% better when a larger vocabulary (4215 tag-word pairs) is used.

(i.e. pruning the search to 1 alternative directly after every word) to one word of lookahead. After one word of lookahead the curves show relatively moderate improvements with each additional word of lookahead, converging to the non-deterministic level, as would be expected.[4] But between zero words of lookahead and one word of lookahead there is a 5.6% absolute improvement in F-measure (versus a 0.9% absolute improvement between one and two words of lookahead). In other words, adding the first word of lookahead results in a 2/3 reduction in the difference between the deterministic and non-deterministic parser's F-measure, while adding subsequent words results in at most a 1/3 reduction per word.

7 Discussion

The large improvement in performance which results from adding the first word of lookahead, as compared to adding the subsequent words, indicates that the first word of lookahead has a qualitatively different effect on deterministic parsing. We believe that one word of lookahead is both necessary and sufficient for a model of deterministic parsing.

The large gain provided by the first word of lookahead indicates that this lookahead is necessary for deterministic parsing. Given the fact the with one word of lookahead the F-measure of the deterministic parser is only 2.7% below the maximum possible, it is unlikely that the family of deterministic parsers assumed here is so sub-optimal that the entire 5.6% improvement gained with one word lookahead is simply the result of compensating for limitations in the choice of this family.

The performance curves in figure 1 also suggest that one word of lookahead is sufficient. We believe the gain provided by more than one word of lookahead is the result of compensating for limitations in the family of deterministic parsers assumed here. Any limitations in this family will result in the deterministic search making choices before the necessary disambiguating information is available, thereby leading to additional errors. As the lookahead increases, some previously mistaken choices will become disambiguated by the additional lookahead information, thereby improving performance. In the limit as lookahead increases, the performance of the deterministic and non-deterministic parsers will become the same, no matter what family of deterministic parsers has been specified. The smooth curve of increasing performance as the lookahead is increased above one word is the type of results we would expect if the lookahead were simply correcting mistakes in this way.

Examples of possible limitations to the family of deterministic parsers assumed here include the choice of the left-corner ordering of parser decisions. The left-corner ordering completely determines when each decision about the phrase structure tree must be made. If the family of deterministic parsers had more flexibility in this ordering, then the optimal deterministic parser could use an ordering which was tailored to the statistics of the data, thereby avoiding being forced to make decisions before sufficient information is available.

8 Conclusions

In this paper we have investigated issues in deterministic parsing by characterizing these issues in terms of the search procedure used by a statistical parser. We use a neural network to estimate the probabilities for an incremental history-based probability model based on left-corner parsing. Using an unconstrained search procedure to try to find the most probable parse according to this probability model (i.e. non-deterministic parsing) results in state-of-the-art accuracy. Deterministic parsing is simulated by allowing the sequence of decisions between two words to be combined into a single parser action, and choosing the best single combined action based on the probability calculated using the basic left-corner probability model. All parses which do not use this chosen action are then pruned from the search. When this pruning is applied directly after each word, there is a large reduction in accuracy (8.3% F-measure) as compared to the non-deterministic search.

Given the pervasive ambiguity in natural language, it is not surprising that this drastic pruning strategy results in a large reduction in accuracy. For this reason, deterministic parsers usually use some form of lookahead. Lookahead gives the parser more information about the sentence at the point when the choice of the next parser action takes place. We simulate the optimal use of k-word lookahead by summing over all partial parses which continue the given partial parse to the point where all k words in the lookahead have been generated.

[4]Note that when the lookahead length is longer than the longest sentence, the deterministic and non-deterministic parsers become equivalent.

When expressed in terms of search, this means that the deterministic pruning is done k words behind a non-deterministic search for the best parse, based on a sum over the partial parses found by the non-deterministic search. When accuracy is plotted as a function of k (figure 1), we found that there is a large increase in accuracy when the first word of lookahead is added (only 2.7% F-measure below non-deterministic search). Further increases in the lookahead length have much less of an impact.

We conclude that the first word of lookahead is necessary for the success of any deterministic parser, but that additional lookahead is probably not necessary. The remaining error created by this model of deterministic parsing is probably best dealt with by investigating other aspect of the model of deterministic parsing assumed here, in particular the strict adherence to the left-corner parsing order.

Despite the need to consider alternatives to the left-corner parsing order, these results do demonstrate that the left-corner parsing strategy proposed is surprisingly good at supporting deterministic parsing. This fact is important in making the non-deterministic search strategy used with this parser tractable. The observations made in this paper could lead to more sophisticated search strategies which further increase the speed of this or similar parsers without significant reductions in accuracy.

References

Christopher M. Bishop. 1995. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, UK.

Rens Bod. 2003. An efficient implementation of a new DOP model. In *Proc. 10th Conf. of European Chapter of the Association for Computational Linguistics*, Budapest, Hungary.

Thorsten Brants and Matthew Crocker. 2000. Probabilistic parsing and psychological plausibility. In *Proceedings of the Eighteenth Conference on Computational Linguistics (COLING-2000)*, Saarbrücken / Luxemburg / Nancy.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. 1st Meeting of North American Chapter of Association for Computational Linguistics*, pages 132–139, Seattle, Washington.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing.* Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

F. Costa, V. Lombardo, P. Frasconi, and G. Soda. 2001. Wide coverage incremental parsing by learning attachment preferences. In *Proc. of the Conf. of the Italian Association for Artificial Intelligence.*

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proc. joint meeting of North American Chapter of the Association for Computational Linguistics and the Human Language Technology Conf.*, pages 103–110, Edmonton, Canada.

Christopher D. Manning and Bob Carpenter. 1997. Probabilistic parsing using left corner language models. In *Proc. Int. Workshop on Parsing Technologies*, pages 147–158.

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

Mitchell Marcus. 1980. *A Theory of Syntactic Recognition for Natural Language.* MIT Press, Cambridge, MA.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proc. Conf. on Empirical Methods in Natural Language Processing*, pages 133–142, Univ. of Pennsylvania, PA.

D.J. Rosenkrantz and P.M. Lewis. 1970. Deterministic left corner parsing. In *Proc. 11th Symposium on Switching and Automata Theory*, pages 139–152.