A Best-First Probabilistic Shift-Reduce Parser

Kenji Sagae and **Alon Lavie**
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
{sagae,alavie}@cs.cmu.edu

Abstract

Recently proposed deterministic classifier-based parsers (Nivre and Scholz, 2004; Sagae and Lavie, 2005; Yamada and Matsumoto, 2003) offer attractive alternatives to generative statistical parsers. Deterministic parsers are fast, efficient, and simple to implement, but generally less accurate than optimal (or nearly optimal) statistical parsers. We present a statistical shift-reduce parser that bridges the gap between deterministic and probabilistic parsers. The parsing model is essentially the same as one previously used for deterministic parsing, but the parser performs a best-first search instead of a greedy search. Using the standard sections of the WSJ corpus of the Penn Treebank for training and testing, our parser has 88.1% precision and 87.8% recall (using automatically assigned part-of-speech tags). Perhaps more interestingly, the parsing model is significantly different from the generative models used by other well-known accurate parsers, allowing for a simple combination that produces precision and recall of 90.9% and 90.7%, respectively.

1 Introduction

Over the past decade, researchers have developed several constituent parsers trained on annotated data that achieve high levels of accuracy. Some of the more popular and more accurate of these approaches to data-driven parsing (Charniak, 2000; Collins, 1997; Klein and Manning, 2002) have been based on generative models that are closely related to probabilistic context-free grammars. Recently, classifier-based dependency parsing (Nivre and Scholz, 2004; Yamada and Matsumoto, 2003) has showed that deterministic parsers are capable of high levels of accuracy, despite great simplicity. This work has led to the development of deterministic parsers for constituent structures as well (Sagae and Lavie, 2005; Tsuruoka and Tsujii, 2005). However, evaluations on the widely used WSJ corpus of the Penn Treebank (Marcus et al., 1993) show that the accuracy of these parsers still lags behind the state-of-the-art.

A reasonable and commonly held assumption is that the accuracy of deterministic classifier-based parsers can be improved if determinism is abandoned in favor of a search over a larger space of possible parses. While this assumption was shown to be true for the parser of Tsuruoka and Tsujii (2005), only a moderate improvement resulted from the addition of a non-greedy search strategy, and overall parser accuracy was still well below that of state-of-the-art statistical parsers.

We present a statistical parser that is based on a shift-reduce algorithm, like the parsers of Sagae and Lavie (2005) and Nivre and Scholz (2004), but performs a best-first search instead of pursuing a single analysis path in deterministic fashion. The parser retains much of the simplicity of deterministic classifier-based parsers, but achieves results that are closer in accuracy to state-of-the-art statistical parsers. Furthermore, a simple combination of the shift-reduce parsing model with an existing generative parsing model produces results with accuracy that surpasses any that of any single (non-reranked) parser tested on the WSJ Penn Treebank, and comes close to the best results obtained with discriminative reranking (Charniak and Johnson, 2005).

2 Parser Description

Our parser uses an extended version of the basic bottom-up shift-reduce algorithm for constituent structures used in Sagae and Lavie's (2005) deterministic parser. For clarity, we will first describe the deterministic version of the algorithm, and then show how it can be extended into a probabilistic algorithm that performs a best-first search.

2.1 A Shift-Reduce Algorithm for Deterministic Constituent Parsing

In its deterministic form, our parsing algorithm is the same single-pass shift-reduce algorithm as the one used in the classifer-based parser of Sagae and Lavie (2005). That algorithm, in turn, is similar to the dependency parsing algorithm of Nivre and Scholz (2004), but it builds a constituent tree and a dependency tree simultaneously. The algorithm considers only trees with unary and binary productions. Training the parser with arbitrary branching trees is accomplished by a simple procedure to transform those trees into trees with at most binary productions. This is done by converting each production with n children, where $n > 2$, into $n - 1$ binary productions. This binarization process is similar to the one described in (Charniak et al., 1998). Additional non-terminal nodes introduced in this conversion must be clearly marked. Transforming the parser's output into arbitrary branching trees is accomplished using the reverse process.

The deterministic parsing algorithm involves two main data structures: a stack S, and a queue W. Items in S may be terminal nodes (part-of-speech-tagged words), or (lexicalized) subtrees of the final parse tree for the input string. Items in W are terminals (words tagged with parts-of-speech) corresponding to the input string. When parsing begins, S is empty and W is initialized by inserting every word from the input string in order, so that the first word is in front of the queue.

The algorithm defines two types of parser actions, shift and reduce, explained below:

- Shift: A shift action consists only of removing (shifting) the first item (part-of-speech-tagged word) from W (at which point the next word becomes the new first item), and placing it on top of S.
- Reduce: Reduce actions are subdivided into unary and binary cases. In a unary reduction, the item on top of S is popped, and a new item is pushed onto S. The new item consists of a tree formed by a non-terminal node with the popped item as its single child. The lexical head of the new item is the same as the lexical head of the popped item. In a binary reduction, two items are popped from S in sequence, and a new item is pushed onto S. The new item consists of a tree formed by a non-terminal node with two children: the first item popped from S is the right child, and the second item is the left child. The lexical head of the new item may be the lexical head of its left child, or the lexical head of its right child.

If S is empty, only a shift action is allowed. If W is empty, only a reduce action is allowed. If both S and W are non-empty, either shift or reduce actions are possible, and the parser must decide whether to shift or reduce. If it decides to reduce, it must also choose between a unary-reduce or a binary-reduce, what non-terminal should be at the root of the newly created subtree to be pushed onto the stack S, and whether the lexical head of the newly created subtree will be taken from the right child or the left child of its root node. Following the work of Sagae and Lavie, we consider the complete set of decisions associated with a reduce action to be part of that reduce action. Parsing terminates when W is empty and S contains only one item, and the single item in S is the parse tree for the input string.

2.2 Shift-Reduce Best-First Parsing

A deterministic shift-reduce parser based on the algorithm described in section 2.1 does not handle ambiguity. By choosing a single parser action at each opportunity, the input string is parsed deterministically, and a single constituent structure is built during the parsing process from beginning to end (no other structures are even considered).

A simple extension to this idea is to eliminate determinism by allowing the parser to choose several actions at each opportunity, creating different paths that lead to different parse trees. This is essentially the difference between deterministic LR parsing (Knuth, 1965) and Generalized-LR parsing (Tomita, 1987; Tomita, 1990). Furthermore, if a probability is assigned to every parser action, the probability of a parse tree can be computed simply as the product of the probabilities of each action in the path that resulted in that parse tree (the derivation of the tree). This produces a probabilistic shift-reduce parser that resembles a generalized probabilistic LR parser (Briscoe and Carroll, 1993), where probabilities are associated with an LR parsing table. In our case, although there is no LR table, the action probabilities are associated with several aspects of the current state of the parser, which to some extent parallel the information contained in an LR table. Instead of having an explicit LR table and pushing LR states onto the stack, the state of the parser is implicitly defined by the configurations of the stack and queue. In a way, there is a parallel between how modern PCFG-like parsers use markov grammars as a distribution that is used to determine the probability of any possible grammar rules, and the way a statistical model is used in our parser to assign a probability to any transition of parser states (instead of a symbolic LR table).

Pursuing every possible sequence of parser actions creates a very large space of actions for even moderately sized sentences. To find the most likely parse tree efficiently according to the probabilistic shift-reduce parsing scheme described so far, we use a best-first strategy. This involves an extension of the deterministic shift-reduce algorithm into a best-first shift-reduce algorithm. To describe this extension, we first introduce a new data structure T_i that represents a parser state, which includes a stack S_i and a queue W_i. In the deterministic algorithm, we would have a single parser state T that contains S and W. The best-first algorithm, on the other hand, has a heap H containing multiple parser states T_1 ... T_n. These states are ordered in the heap according to their probabilities, so that the state with the highest probability is at the top. State probabilities are determined by multiplying the probabilities of each of the actions that resulted in that state. Parser actions are determined from and applied to a parser state T_i popped from the top of H. The parser actions are the same as in the deterministic version of the algorithm. When the item popped from the top of the heap H contains a stack S_i with a single item and an empty queue (in other words, meets the acceptance criteria for the deterministic version of the algorithm), the item on top of S_i is the tree with the highest probability. At that point, parsing terminates if we are searching for the most probable parse. To obtain a list of n-best parses, we simply continue parsing once the first parse tree is found, until either n trees are found, or H is empty.

We note that this approach does not use dynamic programming, and relies only on the best-first search strategy to arrive at the most probable parse efficiently. Without any pruning of the search space, the distribution of probability mass among different possible actions for a parse state has a large impact on the behavior of the search. We do not use any normalization to account for the size (in number of actions) of different derivations when calculating their probabilities, so it may seem that shorter derivations usually have higher probabilities than longer ones, causing the best-first search to approximate a breadth-first search in practice. However, this is not the case if for a given parser state only a few actions (or, ideally, only one action) have high probability, and all other actions have very small probabilities. In this case, only likely derivations would reach the top of the heap, resulting in the desired search behavior. The accuracy of deterministic parsers suggest that this may in fact be the types of probabilities a classifier would produce given features that describe the parser state, and thus the context of the parser action, specifically enough. The experiments described in section 4 support this assumption.

2.3 Classifier-Based Best-First Parsing

To build a parser based on the deterministic algorithm described in section 2.1, a classifier is used to determine parser actions. Sagae and Lavie (2005) built two deterministic parsers this way, one using support vector machines, and one using k-nearest neighbors. In each case, the set of features and classes used with each classifier was the same. Items 1 – 13 in figure 1 shows the features used by Sagae and Lavie. The classes produced by the classifier encode every aspect of a parser action. Classes have one of the following forms:

SHIFT : represents a shift action;

REDUCE-UNARY-*XX* : represents a unary reduce action, where the root of the new subtree pushed onto S is of type *XX* (where *XX* is a non-terminal symbol, typically NP, VP, PP, for example);

REDUCE-LEFT-*XX* : represents a binary reduce action, where the root of the new subtree pushed onto S is of non-terminal type *XX*. Additionally, the head of the new subtree is the same as the head of the left child of the root node;

REDUCE-RIGHT-*XX* : represents a binary reduce action, where the root of the new subtree pushed onto S is of non-terminal type *XX*. Additionally, the head of the new subtree is the same as the head of the right child of the root node.

To implement a parser based on the best-first algorithm, instead of just using a classifier to determine one parser action given a stack and a queue, we need a classification approach that provides us with probabilities for different parser actions associated with a given parser state. One such approach is maximum entropy classification (Berger et al., 1996), which we use in the form of a library implemented by Tsuruoka[1] and used in his classifier-based parser (Tsuruoka and Tsujii, 2005). We used the same classes and the same features as Sagae and Lavie, and an additional feature that represents the previous parser action applied the current parser state (figure 1).

3 Related Work

As mentioned in section 2, our parsing approach can be seen as an extension of the approach of Sagae and Lavie (2005). Sagae and Lavie evaluated their deterministic classifier-based parsing framework using two classifiers: support vector machines (SVM) and k-nearest neighbors (kNN). Although the kNN-based parser performed poorly, the SVM-based parser achieved about 86% precision and recall (or 87.5% using gold-standard POS tags) on the WSJ test section of the Penn Treebank, taking only 11 minutes to parse the test set. Sagae and Lavie's parsing algorithm is similar to the one used by Nivre and Scholz (2004) for deterministic dependency parsing (using kNN). Yamada and Matsumoto (2003) have also presented a deterministic classifier-based (SVM-based) dependency parser, but using a different parsing algorithm, and using only unlabeled dependencies.

Tsuruoka and Tsujii (2005) developed a classifier-based parser that uses the chunk-parsing algorithm and achieves extremely high parsing speed, but somewhat low recall. The algorithm is based on reframing the parsing task as several sequential chunking tasks.

Finally, our parser is in many ways similar to the parser of Ratnaparkhi (1997). Ratnaparkhi's parser uses maximum-entropy models to determine the actions of a parser based to some extent on the shift-reduce framework, and it is also capable of pursuing several paths and returning the top-n highest scoring parses for a sentence. However, in addition to using different features for parsing, Ratnaparkhi's parser uses a different, more complex algorithm. The use of a more involved algorithm allows Ratnaparkhi's parser to work with arbitrary branching trees without the need of the binarization transform employed here. It breaks the usual reduce actions into smaller pieces (CHECK and BUILD), and uses two separate passes (not including the part-of-speech tagging pass) for determining chunks and higher syntactic structures separately. Instead of keeping a stack, the parser makes multiple passes over the input string, like the dependency parsing algorithm used by Yamada and Matsumoto. Our parser, on the other hand, uses a simpler stack-based shift-reduce (LR-like) algorithm for trees with only unary and binary productions.

4 Experiments

We evaluated our classifier-based best-first parser on the Wall Street Journal corpus of the Penn Treebank (Marcus et al., 1993) using the standard split: sections 2-21 were used for training, section 22 was used for development and tuning of parameters and features, and section 23 was used for testing. Every experiment reported here was performed on a Pentium4 3.2GHz with 2GB of RAM.

Each tree in the training set had empty-node and function tag information removed, and the trees were lexicalized using the same head-table rules as in the Collins (1999) parser (these rules were taken from Bikel's (2002) implementation of the Collins parser). The trees were then converted into trees containing only unary and binary productions, using the binarization transform described in section 2. Classifier training instances of features paired with classes (parser actions) were extracted from the trees in the training set, and the total number of training instances was about 1.9 million. It is interesting to note that the procedure of training the best-first parser is identical to the training of a deterministic version of the parser: the deterministic

[1]The SS MaxEnt library is publicly available from http://www-tsujii.is.s.u-tokyo.ac.jp/ tsuruoka/maxent/.

Let:

$S(n)$ denote the nth item from the top of the stack S, and
$W(n)$ denote the nth item from the front of the queue W.

Features:

1. The head-word (and its POS tag) of: $S(0)$, $S(1)$, $S(2)$, $andS(3)$

2. The head-word (and its POS tag) of: $W(0)$, $W(1)$, $W(2)$ and $W(3)$

3. The non-terminal node of the root of: $S(0)$, and $S(1)$

4. The non-terminal node of the left child of the root of: $S(0)$, and $S(1)$

5. The non-terminal node of the right child of the root of: $S(0)$, and $S(1)$

6. The POS tag of the head-word of the left child of the root of: $S(0)$, and $S(1)$

7. The POS tag of the head-word of the right child of the root of: $S(0)$, and $S(1)$

8. The linear distance (number of words apart) between the head-words of $S(0)$ and $S(1)$

9. The number of lexical items (words) that have been found (so far) to be dependents of the head-words of: $S(0)$, and $S(1)$

10. The most recently found lexical dependent of the head-word of $S(0)$ that is to the left of $S(0)$'s head

11. The most recently found lexical dependent of the head-word of $S(0)$ that is to the right of $S(0)$'s head

12. The most recently found lexical dependent of the head-word of $S(1)$ that is to the left of $S(1)$'s head

13. The most recently found lexical dependent of the head-word of $S(1)$ that is to the right of $S(1)$'s head

14. The previous parser action applied to the current parser state

Figure 1: Features used for classification, with features 1 to 13 taken from Sagae and Lavie (2005). The features described in items 1 – 7 are more directly related to the lexicalized constituent trees that are built during parsing, while the features described in items 8 – 13 are more directly related to the dependency structures that are built simultaneously to the constituent structures.

algorithm is simply run over all sentences in the training set, and since the correct trees are known in advance, we can simply record the features and correct parser actions that lead to the construction of the correct tree.

Training the maximum entropy classifier with such a large number (1.9 million) of training instances and features required more memory than was available (the maximum training set size we were able to train with 2GB of RAM was about 200,000 instances), so we employed the training set splitting idea used by Yamada and Matsumoto (2003) and Sagae and Lavie (2005). In our case, we split the training data according to the part-of-speech (POS) tag of the head-word of the item on top of the stack, and trained each split of the training data separately. At run-time, every trained classifier is loaded, and the choice of classifier to use is made by looking at the head-word of the item on top of the stack in the current parser state. The total training time (a single machine was used and each classifier was trained in series) was slightly under nine hours. For comparison, Sagae and Lavie (2005) report that training support vector machines for one-against-all multi-class classification on the same set of features for their deterministic parser took 62 hours, and training a k-nearest neighbors classifier took 11 minutes.

When given perfectly tagged text (gold part-of-speech tags extracted from the Penn Treebank), our parser has labeled constituent precision and recall of 89.40% and 88.79% respectively over all sentences in the test set, and 90.01% and 89.32% over sentences with length of at most 40 words. These results are at the same level of accuracy as those obtained with other state-of-the-art statistical parsers, although still well below the best published results for this test set (Bod, 2003; Charniak and Johnson, 2005). Although the parser is quite accurate, parsing the test set took 41 minutes. By implementing a very simple pruning strategy, the parser can be made much faster. Pruning the search space is done by only adding a new parser state to the heap if its probability is greater than $1/b$ of the probability of the most likely state in the heap that has had the same number of parser actions. By setting b to 50, the parser's accuracy is only affected minimally, and we obtain 89.3% precision and 88.7% recall, while parsing the test set in slightly under 17 minutes and taking less than 60 megabytes of RAM. Under the same conditions, but using automatically assigned part-of-speech tags (at 97.1% accuracy) using the SVMTool tagger (Gimenez and Marquez, 2004), we obtain 88.1% precision and 87.8% recall. It is likely that the deterioration in accuracy is aggravated by the training set splitting scheme based on POS tags.

A deterministic version of our parser, obtained by simply taking the most likely parser action as the only action at each step (in other words, by setting b to 1), has precision and recall of 85.4% and 84.8%, respectively (86.5% and 86.0% using gold-standard POS tags). More interestingly, it parses all 2,416 sentences (more than 50,000 words) in only 46 seconds, 10 times faster than the deterministic SVM parser of Sagae and Lavie (2005). The parser of Tsuruoka and Tsujii (Tsuruoka and Tsujii, 2005) has comparable speed, but we obtain more accurate results. In addition to being fast, our deterministic parser is also lean, requiring only about 25 megabytes of RAM.

A summary of these results is shown in table 1, along with the results obtained with other parsers for comparison purposes. The figures shown in table 1 only include experiments using automatically assigned POS tags. Results obtained with gold-standard POS tags are not shown, since they serve little purpose in a comparison with existing parsers. Although the time figures reflect the performance of each parser at the stated level of accuracy, all of the search-based parsers can trade accuracy for increased speed. For example, the Charniak parser can be made twice as fast at the cost of a 0.5% decrease in precision/recall, or ten times as fast at the cost of a 4% decrease in precision/recall (Roark and Charniak, 2002).

4.1 Reranking with the Probabililstic Shift-Reduce Model

One interesting aspect of having an accurate parsing model that is significantly different from other well-known generative models is that the combination of two accurate parsers may produce even more accurate results. A probabilistic shift-reduce LR-like model, such as the one used in our parser, is different in many ways from a lexicalized PCFG-like model (using markov a grammar), such as those used in the Collins (1999) and Charniak (2000) parsers. In the probabilistic LR model, probabilities are assigned to tree

	Precision	Recall	F-score	Time (min)
Best-First Classifier-Based (this paper)	88.1	87.8	87.9	17
Deterministic (MaxEnt) (this paper)	85.4	84.8	85.1	< 1
Charniak & Johnson (2005)	91.3	90.6	91.0	Unk
Bod (2003)	90.8	90.7	90.7	145*
Charniak (2000)	89.5	89.6	89.5	23
Collins (1999)	88.3	88.1	88.2	39
Ratnaparkhi (1997)	87.5	86.3	86.9	Unk
Tsuruoka & Tsujii (2005): deterministic	86.5	81.2	83.8	< 1*
Tsuruoka & Tsujii (2005): search	86.8	85.0	85.9	2*
Sagae & Lavie (2005)	86.0	86.1	86.0	11*

Table 1: Summary of results on labeled precision and recall of constituents, and time required to parse the test set. We first show results for the parsers described here, then for four of the most accurate or most widely known parsers, for the Ratnaparkhi maximum entropy parser, and finally for three recent classifier-based parsers. For the purposes of direct comparisons, only results obtained with automatically assigned part-of-speech tags are shown (tags are assigned by the parser itself or by a separate part-of-speech tagger). * Times reported by authors running on different hardware.

derivations (not the constituents themselves) based on the sequence of parser shift/reduce actions. PCFG-like models, on the other hand, assign probabilities to the trees directly. With models that differ in such fundamental ways, it is possible that the probabilities assigned to different trees are independent enough that even a very simple combination of the two models may result in increased accuracy.

We tested this hypothesis by using the Charniak (2000) parser in n-best mode, producing the top 10 trees with corresponding probabilities. We then rescored the trees produced by the Charniak parser using our probabilistic LR model, and simply multiplied the probabilities assigned by the Charniak model and our LR model to get a combined score for each tree[2]. On development data this resulted in a 1.3% absolute improvement in f-score over the 1-best trees produced by the Charniak parser. On the test set (WSJ Penn Treebank section 23), this reranking scheme produces precision of 90.9% and recall of 90.7%, for an f-score of 90.8%.

[2]The trees produced by the Charniak parser may include the part-of-speech tags AUX and AUXG, which are not part of the original Penn Treebank tagset. See (Charniak, 2000) for details. These are converted deterministically into the appropriate Penn Treebank verb tags, possibly introducing a small number of minor POS tagging errors. Gold-standard tags or the output of a separate part-of-speech tagger are not used at any point in rescoring the trees.

5 Conclusion

We have presented a best-first classifier-based parser that achieves high levels of precision and recall, with fast parsing times and low memory requirements. One way to view the parser is as an extension of recent work on classifier-based deterministic parsing. It retains the modularity between parsing algorithms and learning mechanisms associated with deterministic parsers, making it simple to understand, implement, and experiment with. Another way to view the parser is as a variant of probabilistic GLR parsers without an explicit LR table.

We have shown that our best-first strategy results in significant improvements in accuracy over deterministic parsing. Although the best-first search makes parsing slower, we have implemented a beam strategy that prunes much of the search space with very little cost in accuracy. This strategy involves a parameter that can be used to control the trade-off between accuracy and speed. At one extreme, the parser is very fast (more than 1,000 words per second) and still moderately accurate (about 85% f-score, or 86% using gold-standard POS tags). This makes it possible to apply parsing to natural language tasks involving very large amounts of text (such as question-answering or information extraction with large corpora). A less aggressive pruning setting results in an f-score of about 88% (or 89%, using gold-standard POS tags), taking 17 minutes to parse the WSJ test set.

Finally, we have shown that by multiplying the probabilities assigned by our maximum entropy shift-reduce model to the probabilities of the 10-best trees produced for each sentence by the Charniak parser, we can rescore the trees to obtain more accurate results than those produced by either model in isolation. This simple combination of the two models produces an f-score of 90.8% for the standard WSJ test set.

Acknowledgements

We thank John Carroll for insightful discussions at various stages of this work, and the reviewers for their detailed comments. This work was supported in part by the National Science Foundation under grant IIS-0414630.

References

A. Berger, S. A. Della Pietra, and V. J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

D. Bikel. 2002. Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proceedings of HLT2002*. San Diego, CA.

R. Bod. 2003. An efficient implementation of a new dop model. In *Proceedings of the European chapter of the 2003 meeting of the Association for Computational Linguistics*. Budapest, Hungary.

E. Briscoe and J. Carroll. 1993. Generalised probabilistic lr parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd meeting of the Association for Computational Linguistics*. Ann Arbor, MI.

Eugene Charniak, Sharon Goldwater, and Mark Johnson. 1998. Edge-based best-first chart parsing. In *Proceedings of the Sixth Workshop on Very Large Corpora*. Montreal, Canada.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 132–139. Seattle, WA.

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23.

M. Collins. 1999. *Head-Driven Models for Natural Language Parsing*. Phd thesis, University of Pennsylvania.

J. Gimenez and L. Marquez. 2004. Svmtool: A general pos tagger generator based on support vector machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*. Lisbon, Portugal.

Dan Klein and Christopher D. Manning. 2002. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*. Vancouver, BC.

D. E. Knuth. 1965. On the translation of languages from left to right. *Information and Control*, 8(6):607–639.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewics. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of english text. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 64–70. Geneva, Switzerland.

Adwait Ratnaparkhi. 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*. Providence, RI.

B. Roark and E. Charniak. 2002. Measuring efficiency in high-accuracy, broad coverage statistical parsing. In *Proceedings of the Efficiency in Large-scale Parsing Systems Workshop at COLING-2000*. Luxembourg.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technologies*. Vancouver, BC.

Masaru Tomita. 1987. An efficient augmented context-free parsing algorithm. *Computational Linguistics*, 13:31–46.

Masaru Tomita. 1990. The generalized lr parser/compiler - version 8.4. In *Proceedings of the International Conference on Computational Linguistics (COLING'90)*, pages 59–63. Helsinki, Finland.

Y. Tsuruoka and K. Tsujii. 2005. Chunk parsing revisited. In *Proceedings of the Ninth International Workshop on Parsing Technologies*. Vancouver, Canada.

H. Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis using support vector machines. In *Proceedings of the Eighth International Workshop on Parsing Technologies*. Nancy, France.