Chunk Parsing Revisited

Yoshimasa Tsuruoka[1,2] **and Jun'ichi Tsujii**[2,3,1]
[1] CREST, JST (Japan Science and Technology Corporation)
[2] Department of Computer Science, University of Tokyo
[3] School of Informatics, University of Manchester
{tsuruoka,tsujii}@is.s.u-tokyo.ac.jp

Abstract

Chunk parsing is conceptually appealing but its performance has not been satisfactory for practical use. In this paper we show that chunk parsing can perform significantly better than previously reported by using a simple sliding-window method and maximum entropy classifiers for phrase recognition in each level of chunking. Experimental results with the Penn Treebank corpus show that our chunk parser can give high-precision parsing outputs with very high speed (14 msec/sentence). We also present a parsing method for searching the best parse by considering the probabilities output by the maximum entropy classifiers, and show that the search method can further improve the parsing accuracy.

1 Introduction

Chunk parsing (Tjong Kim Sang, 2001; Brants, 1999) is a simple parsing strategy both in implementation and concept. The parser first performs chunking by identifying base phrases, and convert the identified phrases to non-terminal symbols. The parser again performs chunking on the updated sequence and convert the newly recognized phrases into non-terminal symbols. The parser repeats this procedure until there are no phrases to be chunked. After finishing these chunking processes, we can reconstruct the complete parse tree of the sentence from the chunking results.

Although the conceptual simplicity of chunk parsing is appealing, satisfactory performance for practical use has not yet been achieved with this parsing strategy. Sang achieved an f-score of 80.49 on the Penn Treebank by using the IOB tagging method for each level of chunking (Tjong Kim Sang, 2001). However, there is a very large gap between their performance and that of widely-used practical parsers (Charniak, 2000; Collins, 1999).

The performance of chunk parsing is heavily dependent on the performance of phrase recognition in each level of chunking. We show in this paper that the chunk parsing strategy is indeed appealing in that it can give considerably better performance than previously reported by using a different approach for phrase recognition and that it enables us to build a very fast parser that gives high-precision outputs.

This advantage could open up the possibility of using full parsers for large-scale information extraction from the Web corpus and real-time information extraction where the system needs to analyze the documents provided by the users on run-time.

This paper is organized as follows. Section 2 introduces the overall chunk parsing strategy employed in this work. Section 3 describes the sliding-window based method for identifying chunks. Two filtering methods to reduce the computational cost are presented in sections 4 and 5. Section 6 explains the maximum entropy classifier and the feature set. Section 7 describes methods for searching the best parse. Experimental results on the Penn Treebank corpus are given in Section 8. Section 10 offers some concluding remarks.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 133–140,
Vancouver, October 2005.

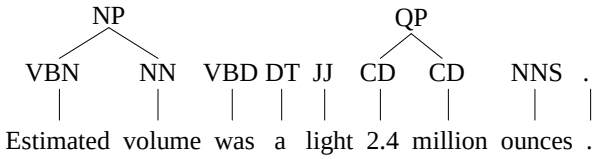

Figure 1: Chunk parsing, the 1st iteration.

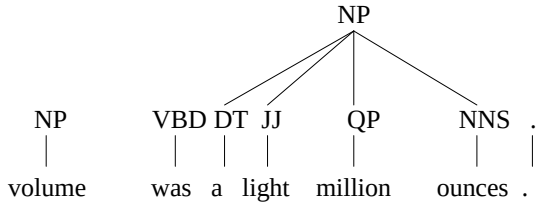

Figure 2: Chunk parsing, the 2nd iteration.

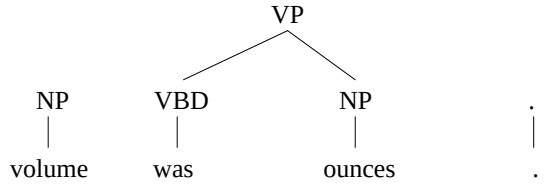

Figure 3: Chunk parsing, the 3rd iteration.

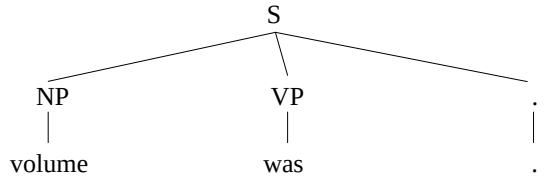

Figure 4: Chunk parsing, the 4th iteration.

2 Chunk Parsing

For the overall strategy of chunk parsing, we follow the method proposed by Sang (Tjong Kim Sang, 2001). Figures 1 to 4 show an example of chunk parsing. In the first iteration, the chunker identifies two base phrases, (NP Estimated volume) and (QP 2.4 million), and replaces each phrase with its non-terminal symbol and head. The head word is identified by using the head-percolation table (Magerman, 1995). In the second iteration, the chunker identifies (NP a light million ounces) and converts this phrase into NP. This chunking procedure is repeated until the whole sentence is chunked at the fourth iteration, and the full parse tree is easily recovered from the chunking history.

This parsing strategy converts the problem of full parsing into smaller and simpler problems, namely, chunking, where we only need to recognize flat structures (base phrases). Sang used the IOB tagging method proposed by Ramshow(Ramshaw and Marcus, 1995) and memory-based learning for each level of chunking and achieved an f-score of 80.49 on the Penn Treebank corpus.

3 Chunking with a sliding-window approach

The performance of chunk parsing heavily depends on the performance of each level of chunking. The popular approach to this shallow parsing is to convert the problem into a tagging task and use a variety of machine learning techniques that have been developed for sequence labeling problems such as Hidden Markov Models, sequential classification with SVMs (Kudo and Matsumoto, 2001), and Conditional Random Fields (Sha and Pereira, 2003).

One of our claims in this paper is that we should not convert the chunking problem into a tagging task. Instead, we use a classical sliding-window method for chunking, where we consider all subsequences as phrase candidates and classify them with a machine learning algorithm. Suppose, for example, we are about to perform chunking on the sequence in Figure 4.

NP-volume VBD-was .-.

We consider the following sub sequences as the phrase candidates in this level of chunking.

1. (NP-volume) VBD-was .-.
2. NP-volume (VBD-was) .-.
3. NP-volume VBD-was (.-.)
4. (NP-volume VBD-was) .-.
5. NP-volume (VBD-was .-.)
6. (NP-volume VBD-was .-.)

The merit of taking the sliding window approach is that we can make use of a richer set of features on recognizing a phrase than in the sequential labeling

approach. We can define arbitrary features on the target candidate (e.g. the whole sequence of non-terminal symbols of the target) and the surrounding context, which are, in general, not available in sequential labeling approaches.

We should mention here that there are some other modeling methods for sequence labeling which allow us to define arbitrary features on the target phrase. Semi-markov conditional random fields (Semi-CRFs) are one of such modeling methods (Sarawagi and Cohen, 2004). Semi-CRFs could give better performance than the sliding-window approach because they can incorporate features on other phrase candidates on the same level of chunking. However, they require additional computational resources for training and parsing, and the use of Semi-CRFs is left for future work.

The biggest disadvantage of the sliding window approach is the cost for training and parsing. Since there are $n(n+1)/2$ phrase candidates when the length of the sequence is n, a naive application of machine learning easily leads to prohibitive consumption of memory and time.

In order to reduce the number of phrase candidates to be considered by machine learning, we introduce two filtering phases into training and parsing. One is done by a rule dictionary. The other is done by a naive Bayes classifier.

4 Filtering with the CFG Rule Dictionary

We use an idea that is similar to the method proposed by Ratnaparkhi (Ratnaparkhi, 1996) for part-of-speech tagging. They used a *Tag Dictionary*, with which the tagger considers only the tag-word pairs that appear in the training sentences as the candidate tags.

A similar method can be used for reducing the number of phrase candidates. We first construct a rule dictionary consisting of all the CFG rules used in the training data. In both training and parsing, we filter out all the sub-sequences that do not match any of the entry in the dictionary.

4.1 Normalization

The rules used in the training data do not cover all the rules in unseen sentences. Therefore, if we take a naive filtering method using the rule dictionary, we substantially lose recall in parsing unseen data.

Original Symbol	Normalized Symbol
NNP, NNS, NNPS, PRP	NN
RBR, RBS	RB
JJR, JJS, PRP$	JJ
VBD, VBZ	VBP
:	,
", "	*NULL*

Table 1: Normalizing preterminals.

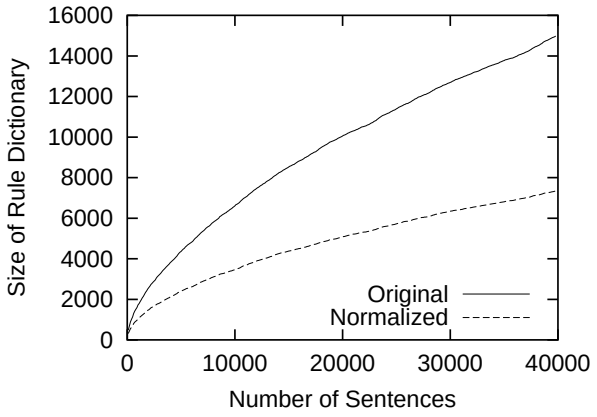

Figure 5: Number of sentences vs the size of the rule dictionary..

To alleviate the problem of the coverage of rules, we conduct normalization of the rules. We first convert preterminal symbols into equivalent sets using the conversion table provided in Table 1. This conversion reduces the sparseness of the rules.

We further normalize the Right-Hand-Side (RHS) of the rules with the following heuristics.

- "X CC X" is converted to "X".
- "X , X" is converted to "X".

Figure 5 shows the effectiveness of this normalization method. The figure illustrates how the number of rules increases in the rule dictionary as we add training sentences. Without the normalization, the number of rules continues to grow rapidly even when the entire training set is read. The normalization methods reduce the growing rate, which considerably alleviates the sparseness problem (i.e. the problems of unknown rules).

5 Filtering with the Naive Bayes classifier

Although the use of the rule dictionary significantly reduced the number of phrase candidates, we still found it difficult to train the parser using the entire training set when we used a rich set of features.

To further reduce the cost required for training and parsing, we propose to use a naive Bayes classifier for filtering the candidates. A naive Bayes classifier is simple and requires little storage and computational cost.

We construct a binary naive Bayes classifier for each phrase type using the entire training data. We considered the following information as the features.

- The Right-Hand-Side (RHS) of the CFG rule
- The left-adjacent nonterminal symbol.
- The right-adjacent nonterminal symbol.

By assuming the conditional independence among the features, we can compute the probability for filtering as follows:

$$\begin{aligned} P(y|c,l,r) &= \frac{P(c,l,r|y)P(y)}{P(c,l,r)} \\ &= \frac{P(c|y)P(l|y)P(r|y)P(y)}{\sum_y P(c|y)P(l|y)P(r|y)P(y)}, \end{aligned}$$

where y is a binary output indicating whether the candidate is a phrase of the target type or not, c is the RHS of the CFG rule, l is the symbol on the left, and r is the symbol on the right. We used the Laplace smoothing method for computing each probability. Note that the information about the result of the rule application, i.e., the LHS symbol, is considered in this filtering scheme because different naive Bayes classifiers are used for different LHS symbols (phrase types).

Table 2 shows the filtering performance in training with sections 02-21 on the Penn Treebank. We set the threshold probability for filtering to be 0.0001 for the experiments reported in this paper. The naive Bayes classifiers effectively reduced the number of candidates with little positive samples that were wrongly filtered out.

6 Phrase Recognition with a Maximum Entropy Classifier

For the candidates which are not filtered out in the above two phases, we perform classification with maximum entropy classifiers (Berger et al., 1996).

We construct a binary classifier for each type of phrases using the entire training set. The training samples for maximum entropy consist of the phrase candidates that have not been filtered out by the CFG rule dictionary and the naive Bayes classifier.

One of the merits of using a maximum entropy classifier is that we can obtain a probability from the classifier in each decision. The probability of each decision represents how likely the candidate is a correct chunk. We accept a chunk only when the probability is larger than the predefined threshold. With this thresholding scheme, we can control the trade-off between precision and recall by changing the threshold value.

Regularization is important in maximum entropy modeling to avoid overfitting to the training data. For this purpose, we use the maximum entropy modeling with inequality constraints (Kazama and Tsujii, 2003). This modeling has one parameter to tune as in Gaussian prior modeling. The parameter is called the *width factor*. We set this parameter to be 1.0 throughout the experiments. For numerical optimization, we used the Limited-Memory Variable-Metric (LMVM) algorithm (Benson and Moré, 2001).

6.1 Features

Table 3 lists the features used in phrase recognition with the maximum entropy classifier. Information about the adjacent non-terminal symbols is important. We use unigrams, bigrams, and trigrams of the adjacent symbols. Head information is also useful. We use unigrams and bigrams of the neighboring heads. The RHS of the CFG rule is also informative. We use the features on RHSs combined with symbol features.

7 Searching the best parse

7.1 Deterministic parsing

The deterministic version of chunk parsing is straight-forward. All we need to do is to repeat chunking until there are no phrases to be chunked.

Symbol	# candidates	# remaining candidates	# positives	# false negative
ADJP	4,043,409	1,052,983	14,389	53
ADVP	3,459,616	1,159,351	19,765	78
NP	7,122,168	3,935,563	313,042	117
PP	3,889,302	1,181,250	94,568	126
S	3,184,827	1,627,243	95,305	99
VP	4,903,020	2,013,229	145,878	144

Table 2: Effectiveness of the naive Bayes filtering on some representative nonterminals.

Symbol Unigrams	s_{i-1}, s_{j+1}
Symbol Bigrams	$s_i s_j$, $s_{i-2}s_{i-1}$, $s_{i-1}s_{j+1}$, $s_{j+1}s_{j+2}$
Symbol Trigrams	$s_{i-3}s_{i-2}s_{i-1}$, $s_{i-2}s_{i-1}s_{j+1}$, $s_{i-1}s_{j+1}s_{j+2}$, $s_{j+1}s_{j+2}s_{j+3}$
Head Unigrams	h_{i-1}, h_{j+1}
Head Bigrams	$h_{i-2}h_{i-1}$, $h_{i-1}s_{i+1}$, $h_{i+1}h_{i+2}$
Symbol-Head Unigrams	$s_i h_i$, $s_j h_j$, $s_k h_k (k \in i...j)$
CFG Rule	RHS
CFG Rule + Symbol Unigram	$s_{i-1}RHS$, $s_{j+1}RHS$
CFG Rule + Symbol Bigram	$s_{i-1}s_{j+1}RHS$

Table 3: Feature templates used in chunking. s_i and s_j represent the non-terminal symbols at the beginning and the ending of the target phrase respectively. h_i and h_j represent the head at the beginning and the ending of the target phrase respectively. RHS represents the Right-Hand-Side of the CFG rule.

If the maximum entropy classifiers give contradictory chunks in each level of chunking, we choose the chunk which has a larger probability than the other ones.

7.2 Parsing with search

We tried to perform searching in chunk parsing in order to investigate whether or not extra effort of searching gives a gain in parsing performance.

The problem is that because the modeling of our chunk parsing provides no explicit probabilistic distribution over the entire parse tree, there is no decisive way to properly evaluate the correctness of each parse. Nevertheless, we can consider the following score on each parse tree.

$$score = \prod_{i \in parse} P_i, \qquad (1)$$

where P_i is the probability of a phrase given by the maximum entropy classifier.

Because exploring all the possibilities of chunking requires prohibitive computational cost, we reduce the search space by focusing only on "uncertain" chunk candidates for the search. In each level of chunking, the chunker provides chunks with their probabilities. We consider only the chunks whose probabilities are within the predefined margin from 0.5. In other words, the chunks whose probabilities are larger than $(0.5 + margin)$ are considered as assured chunks, and thus are fixed when we generate alternative hypotheses of chunking. The chunks whose probabilities are smaller than $(0.5 - margin)$ are simply ignored.

We generate alternative hypotheses in each level of chunking, and search the best parse in a depth-first manner.

7.3 Iterative parsing

We also tried an *iterative parsing* strategy, which was successfully used in probabilistic HPSG parsing (Ninomiya et al., 2005). The parsing strategy is simple. The parser starts with a very low margin and tries to find a successful parse. If the parser cannot find a successful parse, then it increases the margin by a certain step and tries to parse with the wider margin.

8 Experiments

We ran parsing experiments using the Penn Treebank corpus, which is widely used for evaluating parsing algorithms. The training set consists of sections 02-21. We used section 22 as the development data, with which we tuned the feature set and parameters for parsing. The test set consists of section 23 and we report the performance of the parser on the set.

We used the *evalb* script provided by Sekine and Collins for evaluating the labeled recall/precision (LR/LP) of the parser outputs [1]. All the experiments were carried out on a server having a 2.6 GHz AMD Opteron CPU and 16GB memory.

8.1 Speed and Accuracy

First, we show the performance that achieved by deterministic parsing. Table 4 shows the results. We parsed all the sentences in section 23 using gold-standard part-of-speech (POS) tags. The trade-off between precision and recall can be controlled by changing the threshold for recognizing chunks. The fifth row gives the performance achieved with the default threshold (=0.5), where the precision is over 90% but the recall is low (75%). By lowering the threshold, we can improve the recall up to around 81% with 2% loss of precision. The best f-score is 85.06.

The parsing speed is very high. The parser takes only about 34 seconds to parse the entire section. Since this section contains 2,416 sentences, the average time required for parsing one sentence is 14 msec. The parsing speed slightly dropped when we used a lower threshold (0.1).

Table 5 shows the performance achieved when we used the search algorithm described in Section 7.2. We limited the maximum number of the nodes in the search space to 100 because further increase of the nodes had shown little improvement in parsing accuracy.

The search algorithm significantly boosted the precisions and recalls and achieved an f-score of 86.52 when the margin was 0.3. It should be noted that we obtain no gain when we use a tight margin. We need to consider phrases having low probabilities in order for the search to work.

[1] We used the parameter file "COLLINS.prm"

Threshold	LR	LP	F-score	Time (sec)
0.9	47.61	96.43	63.75	30.6
0.8	58.06	94.29	71.87	32.4
0.7	65.33	92.82	76.69	33.2
0.6	70.89	91.67	79.95	33.2
0.5	75.38	90.71	82.34	34.5
0.4	79.11	89.87	84.15	34.2
0.3	80.95	88.80	84.69	33.9
0.2	82.59	87.69	85.06	33.6
0.1	82.32	85.02	83.65	46.9

Table 4: Parsing performance on section 23 (all sentences, gold-standard POS tags) with the deterministic algorithm.

Margin	LR	LP	F-score	Time (sec)
0.0	75.65	90.81	82.54	41.2
0.1	79.63	90.16	84.57	74.4
0.2	82.70	89.57	86.00	94.8
0.3	84.60	88.53	86.52	110.2
0.4	84.91	86.99	85.94	116.3

Table 5: Parsing performance on section 23 (all sentences, gold-standard POS tags) with the search algorithm.

One of the advantages of our chunk parser is its parsing speed. For comparison, we show the trade-off between parsing time and performance in Collins parser (Collins, 1999) and our chunk parser in Figure 6. Collins parser allows the user to change the size of the beam in parsing. We used Model-2 because it gave better performance than Model-3 when the beam size was smaller than 1000. As for the chunk parser, we controlled the trade-off by changing the maximum number of nodes in the search. The uncertainty margin for chunk recognition was 0.3. Figure 6 shows that Collins parser clearly outperforms our chunk parser when the beam size is large. However, the performance significantly drops with a smaller beam size. The break-even point is at around 200 sec (83 msec/sentence).

8.2 Comparison with previous work

Table 6 summarizes our parsing performance on section 23 together with the results of previous studies. In order to make the results directly comparable, we produced POS tags as the input of our parsers by using a POS tagger (Tsuruoka and Tsujii, 2005) which was trained on sections 0-18 in the WSJ corpus.

The table also shows the performance achieved

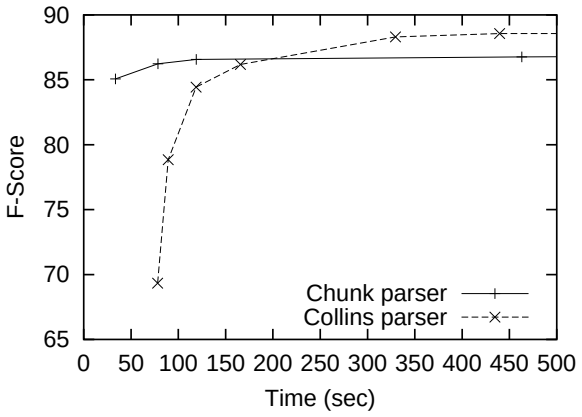

Figure 6: Time vs F-score on section 23. The x-axis represents the time required to parse the entire section. The time required for making a hash table in Collins parser is excluded.

	LR	LP	F-score
Ratnaparkhi (1997)	86.3	87.5	86.9
Collins (1999)	88.1	88.3	88.2
Charniak (2000)	89.6	89.5	89.5
Kudo (2005)	89.3	89.6	89.4
Sang (2001)	78.7	82.3	80.5
Deterministic (tagger-POSs)	81.2	86.5	83.8
Deterministic (gold-POSs)	82.6	87.7	85.1
Search (tagger-POSs)	83.2	87.1	85.1
Search (gold-POSs)	84.6	88.5	86.5
Iterative Search (tagger-POSs)	85.0	86.8	85.9
Iterative Search (gold-POSs)	86.2	88.0	87.1

Table 6: Comparison with other work. Parsing performance on section 23 (all sentences).

with the iterative parsing method presented in section 7.3. Our chunk parser achieved an f-score of 83.8 with the deterministic parsing methods using the POS-tagger tags. This f-score is better than that achieved by the previous study on chunk parsing by 3.3 points (Tjong Kim Sang, 2001). The search algorithms gave an additional 1.3 point improvement. Finally, the iterative parsing method achieved an f-score of 85.9.

Although our chunk parser showed considerably better performance than the previous study on chunk parsing, the performance is still significantly lower than those achieved by state-of-the-art parsers.

9 Discussion

There is a number of possible improvements in our chunk parser. We used a rule dictionary to reduce the cost required for training and parsing. However, the use of the rule dictionary makes the parser fail to identify a correct phrase if the phrase is not contained in the rule dictionary. Although we applied some normalization techniques in order to alleviate this problem, we have not completely solved the problem. Figure 5 indicates that still we will face unknown rules even when we have constructed the rule dictionary using the whole training data (note that the dotted line does not saturate).

Additional feature sets for the maximum entropy classifiers could improve the performance. The bottom-up parsing strategy allows us to use information about sub-trees that have already been constructed. We thus do not need to restrict ourselves to use only head-information of the partial parses. Since many researchers have reported that information on partial parse trees plays an important role for achieving high performance (Bod, 1992; Collins and Duffy, 2002; Kudo et al., 2005), we expect that additional features will improve the performance of chunk parsing.

Also, the methods for searching the best parse presented in sections 7.2 and 7.3 have much room for improvement. the search method does not have the device to avoid repetitive computations on the same nonterminal sequence in parsing. A chart-like structure which effectively stores the partial parse results could enable the parser to explore a broader search space and produce better parses.

Our chunk parser exhibited a considerable improvement in parsing accuracy over the previous study on chunk parsing. However, the reason is not completely clear. We believe that the sliding window approach, which enabled us to exploit a richer set of features than the so-called IOB approach, was the main contributer of the better performance. However, the combination of the IOB approach and a state-of-the-art machine learning algorithm such as support vector machines could produce a similar level of performance. In our preliminary experiments, the IOB tagging method with maximum entropy markov models has not yet achieved a comparable performance to the sliding window method.

10 Conclusion

In this paper we have shown that chunk parsing can perform significantly better than previously reported by using a simple sliding-window method and maximum entropy classifiers in each level of chunking. Experimental results on the Penn Treebank corpus show that our chunk parser can give high-precision parsing outputs with very high speed (14 msec/sentence). We also show that searching can improve the performance and the f-score reaches 85.9.

Although there is still a large gap between the accuracy of our chunk parser and the state-of-the-art, our parser can produce better f-scores than a widely-used parser when the parsing speed is really needed. This could open up the possibility of using full-parsing for large-scale information extraction.

References

Steven J. Benson and Jorge Moré. 2001. A limited-memory variable-metric algorithm for bound-constrained minimization. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory. ANL/MCS-P909-0901.

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

Rens Bod. 1992. Data oriented parsing. In *Proceedings of COLING 1992*.

Thorsten Brants. 1999. Cascaded markov models. In *Proceedings of EACL 1999*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL 2000*, pages 132–139.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of ACL 2002*.

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

Jun'ichi Kazama and Jun'ichi Tsujii. 2003. Evaluation and extension of maximum entropy models with inequality constraints. In *Proceedings of EMNLP 2003*.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of NAACL 2001*.

Taku Kudo, Jun Suzuki, and Hideki Isozaki. 2005. Boosting-based parse reranking with subtree features. In *Proceedings of ACL 2005*.

David M. Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of ACL 1995*, pages 276–283.

Takashi Ninomiya, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic hpsg parsing. In *Proceedings of IWPT 2005*.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP 1996*, pages 133–142.

Adwait Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of EMNLP 1997*, pages 1–10.

Sunita Sarawagi and William W. Cohen. 2004. Semi-markov conditional random fields for information extraction. In *Proceedings of ICML 2004*.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL 2003*.

Erik Tjong Kim Sang. 2001. Transforming a chunker to a parser. In J. Veenstra W. Daelemans, K. Sima'an and J. Zavrel, editors, *Computational Linguistics in the Netherlands 2000*, pages 177–188. Rodopi.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of HLT/EMNLP 2005*.