Fast Full Parsing by Linear-Chain Conditional Random Fields

Yoshimasa Tsuruoka[†‡] **Jun'ichi Tsujii**[†‡*] **Sophia Ananiadou**[†‡]
[†] School of Computer Science, University of Manchester, UK
[‡] National Centre for Text Mining (NaCTeM), UK
[*] Department of Computer Science, University of Tokyo, Japan
{yoshimasa.tsuruoka,j.tsujii,sophia.ananiadou}@manchester.ac.uk

Abstract

This paper presents a chunking-based discriminative approach to full parsing. We convert the task of full parsing into a series of chunking tasks and apply a conditional random field (CRF) model to each level of chunking. The probability of an entire parse tree is computed as the product of the probabilities of individual chunking results. The parsing is performed in a bottom-up manner and the best derivation is efficiently obtained by using a depth-first search algorithm. Experimental results demonstrate that this simple parsing framework produces a fast and reasonably accurate parser.

1 Introduction

Full parsing analyzes the phrase structure of a sentence and provides useful input for many kinds of high-level natural language processing such as summarization (Knight and Marcu, 2000), pronoun resolution (Yang et al., 2006), and information extraction (Miyao et al., 2008). One of the major obstacles that discourage the use of full parsing in large-scale natural language processing applications is its computational cost. For example, the MEDLINE corpus, a collection of abstracts of biomedical papers, consists of 70 million sentences and would require more than two years of processing time if the parser needs one second to process a sentence.

Generative models based on lexicalized PCFGs enjoyed great success as the machine learning framework for full parsing (Collins, 1999; Charniak, 2000), but recently discriminative models attract more attention due to their superior accuracy (Charniak and Johnson, 2005; Huang, 2008) and adaptability to new grammars and languages (Buchholz and Marsi, 2006).

A traditional approach to discriminative full parsing is to convert a full parsing task into a series of classification problems. Ratnaparkhi (1997) performs full parsing in a bottom-up and left-to-right manner and uses a maximum entropy classifier to make decisions to construct individual phrases. Sagae and Lavie (2006) use the shift-reduce parsing framework and a maximum entropy model for local classification to decide parsing actions. These approaches are often called *history-based* approaches.

A more recent approach to discriminative full parsing is to treat the task as a single structured prediction problem. Finkel et al. (2008) incorporated rich local features into a tree CRF model and built a competitive parser. Huang (2008) proposed to use a parse forest to incorporate non-local features. They used a perceptron algorithm to optimize the weights of the features and achieved state-of-the-art accuracy. Petrov and Klein (2008) introduced latent variables in tree CRFs and proposed a caching mechanism to speed up the computation.

In general, the latter whole-sentence approaches give better accuracy than history-based approaches because they can better trade off decisions made in different parts in a parse tree. However, the whole-sentence approaches tend to require a large computational cost both in training and parsing. In contrast, history-based approaches are less computationally intensive and usually produce fast parsers.

In this paper, we present a history-based parser using CRFs, by treating the task of full parsing as a series of chunking problems where it recognizes chunks in a flat input sequence. We use the linear-

Proceedings of the 12th Conference of the European Chapter of the ACL, pages 790–798,
Athens, Greece, 30 March – 3 April 2009.

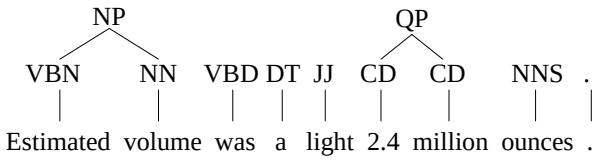

Figure 1: Chunking, the first (base) level.

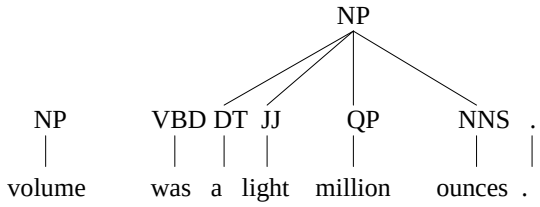

Figure 2: Chunking, the 2nd level.

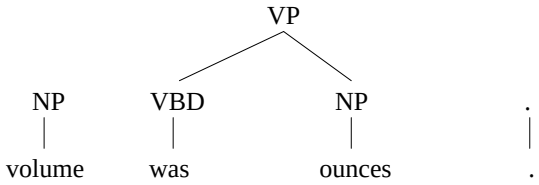

Figure 3: Chunking, the 3rd level.

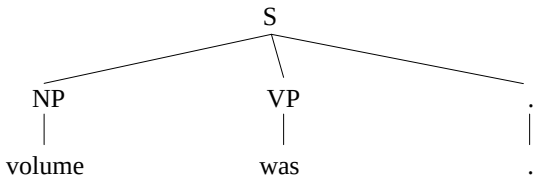

Figure 4: Chunking, the 4th level.

chain CRF model to perform chunking.

Although our parsing model falls into the category of history-based approaches, it is one step closer to the whole-sentence approaches because the parser uses a whole-sequence model (i.e. CRFs) for individual chunking tasks. In other words, our parser could be located somewhere between traditional history-based approaches and whole-sentence approaches. One of our motivations for this work was that our parsing model may achieve a better balance between accuracy and speed than existing parsers.

It is also worth mentioning that our approach is similar in spirit to supertagging for parsing with lexicalized grammar formalisms such as CCG and HPSG (Clark and Curran, 2004; Ninomiya et al., 2006), in which significant speed-ups for parsing time are achieved.

In this paper, we show that our approach is indeed appealing in that the parser runs very fast and gives competitive accuracy. We evaluate our parser on the standard data set for parsing experiments (i.e. the Penn Treebank) and compare it with existing approaches to full parsing.

This paper is organized as follows. Section 2 presents the overall chunk parsing strategy. Section 3 describes the CRF model used to perform individual chunking steps. Section 4 describes the depth-first algorithm for finding the best derivation of a parse tree. The part-of-speech tagger used in the parser is described in section 5. Experimental results on the Penn Treebank corpus are provided in Section 6. Section 7 discusses possible improvements and extensions of our work. Section 8 offers some concluding remarks.

2 Full Parsing by Chunking

This section describes the parsing framework employed in this work.

The parsing process is conceptually very simple. The parser first performs chunking by identifying base phrases, and converts the identified phrases to non-terminal symbols. It then performs chunking for the updated sequence and converts the newly recognized phrases into non-terminal symbols. The parser repeats this process until the whole sequence is chunked as a sentence

Figures 1 to 4 show an example of a parsing process by this framework. In the first (base) level, 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[1]. In the second level, the chunker identifies a noun phrase, (NP a light million ounces), and converts it into NP. This process is repeated until the whole sentence is chunked at the fourth level. The full parse tree is recovered from the chunking history in a straightforward way.

This idea of converting full parsing into a series of chunking tasks is not new by any means—the history of this kind of approach dates back to 1950s (Joshi and Hopely, 1996). More recently, Brants (1999) used a cascaded Markov model to parse German text. Tjong Kim Sang (2001) used the IOB tagging method to represent chunks and memory-based learning, and achieved an f-score of 80.49 on the WSJ corpus. Tsuruoka and Tsujii (2005) improved upon their approach by using

[1]The head word is identified by using the head-percolation table (Magerman, 1995).

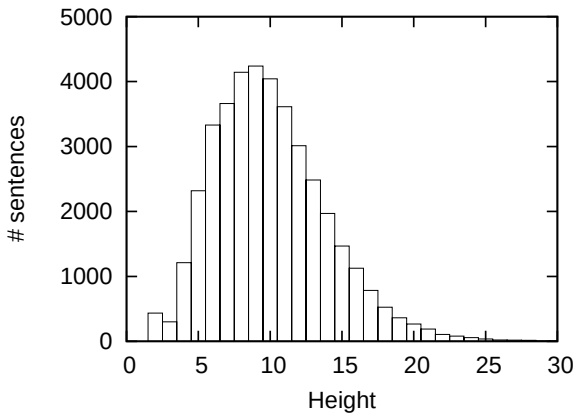

Figure 5: Distribution of tree height in WSJ sections 2-21.

a maximum entropy classifier and achieved an f-score of 85.9. However, there is still a large gap between the accuracy of chunking-based parsers and that of widely-used practical parsers such as Collins parser and Charniak parser (Collins, 1999; Charniak, 2000).

2.1 Heights of Trees

A natural question about this parsing framework is how many levels of chunking are usually needed to parse a sentence. We examined the distribution of the heights of the trees in sections 2-21 of the Wall Street Journal (WSJ) corpus. The result is shown in Figure 5. Most of the sentences have less than 20 levels. The average was 10.0, which means we need to perform, on average, 10 chunking tasks to obtain a full parse tree for a sentence if the parsing is performed in a deterministic manner.

3 Chunking with CRFs

The accuracy of chunk parsing is highly dependent on the accuracy of each level of chunking. This section describes our approach to the chunking task.

A common approach to the chunking problem is to convert the problem into a sequence tagging task by using the "BIO" (B for beginning, I for inside, and O for outside) representation. For example, the chunking process given in Figure 1 is expressed as the following BIO sequences.

B-NP I-NP O O O B-QP I-QP O O

This representation enables us to use the linear-chain CRF model to perform chunking, since the task is simply assigning appropriate labels to a sequence.

3.1 Linear Chain CRFs

A linear chain CRF defines a single log-linear probabilistic distribution over all possible tag sequences $\mathbf{y}$ for the input sequence $\mathbf{x}$:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(t, y_t, y_{t-1}, \mathbf{x}),$$

where $f_k(t, y_t, y_{t-1}, \mathbf{x})$ is typically a binary function indicating the presence of feature k, λ_k is the weight of the feature, and $Z(X)$ is a normalization function:

$$Z(\mathbf{x}) = \sum_{y} \exp \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(t, y_t, y_{t-1}, \mathbf{x}).$$

This model allows us to define features on states and edges combined with surface observations.

The weights of the features are determined in such a way that they maximize the conditional log-likelihood of the training data:

$$\mathcal{L}_\lambda = \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) + R(\lambda),$$

where $R(\lambda)$ is introduced for the purpose of *regularization* which prevents the model from overfitting the training data. The L1 or L2 norm is commonly used in statistical natural language processing (Gao et al., 2007). We used L1-regularization, which is defined as

$$R(\lambda) = \frac{1}{C} \sum_{k=1}^{K} |\lambda_k|,$$

where C is the meta-parameter that controls the degree of regularization. We used the OWL-QN algorithm (Andrew and Gao, 2007) to obtain the parameters that maximize the L1-regularized conditional log-likelihood.

3.2 Features

Table 1 shows the features used in chunking for the base level. Since the task is basically identical to shallow parsing by CRFs, we follow the feature sets used in the previous work by Sha and Pereira (2003). We use unigrams, bigrams, and trigrams of part-of-speech (POS) tags and words.

The difference between our CRF chunker and that in (Sha and Pereira, 2003) is that we could not use second-order CRF models, hence we could not use trigram features on the BIO states. We

Symbol Unigrams	$s_{-2},\ s_{-1},\ s_0,\ s_{+1},\ s_{+2}$
Symbol Bigrams	$s_{-2}s_{-1},\ s_{-1}s_0,\ s_0s_{+1},\ s_{+1}s_{+2}$
Symbol Trigrams	$s_{-3}s_{-2}s_{-1},\ s_{-2}s_{-1}s_0,\ s_{-1}s_0s_{+1},\ s_0s_{+1}s_{+2},\ s_{+1}s_{+2}s_{+3}$
Word Unigrams	$h_{-2},\ h_{-1},\ h_0,\ h_{+1},\ h_{+2}$
Word Bigrams	$h_{-2}h_{-1},\ h_{-1}h_0,\ h_0h_{+1},\ h_{+1}h_{+2}$
Word Trigrams	$h_{-1}h_0h_{+1}$

Table 1: Feature templates used in the base level chunking. s represents a terminal symbol (i.e. POS tag) and the subscript represents a relative position. h represents a word.

found that using second order CRFs in our task was very difficult because of the computational cost. Recall that the computational cost for CRFs is quadratic to the number of possible states. In our task, we need to consider the states for all non-terminal symbols, whereas their work is only concerned with noun phrases.

Table 2 shows feature templates used in the non-base levels of chunking. In the non-base levels of chunking, we can use a richer set of features than the base-level chunking because the chunker has access to the information about the partial trees that have been already created. In addition to the features listed in Table 1, the chunker looks into the daughters of the current non-terminal symbol and use them as features. It also uses the words and POS tags around the edges of the region covered by the current non-terminal symbol. We also added a special feature to better capture PP-attachment. The chunker looks at the head of the second daughter of the prepositional phrase to incorporate the semantic head of the phrase.

4 Searching for the Best Parse

The probability for an entire parse tree is computed as the product of the probabilities output by the individual CRF chunkers:

$$score = \prod_{i=0}^{h} p(\mathbf{y_i}|\mathbf{x_i}), \qquad (1)$$

where i is the level of chunking and h is the height of the tree. The task of full parsing is then to choose the series of chunking results that maximizes this probability.

It should be noted that there are cases where different derivations (chunking histories) lead to the same parse tree (i.e. phrase structure). Strictly speaking, therefore, what we describe here as the probability of a parse tree is actually the probability of a single derivation. The probabilities of the derivations should then be marginalized over to produce the probability of a parse tree, but in this paper we ignore this effect and simply focus only on the best derivation.

We use a depth-first search algorithm to find the highest probability derivation. Figure 6 shows the algorithm in pseudo-code. The parsing process is implemented with a recursive function. In each level of chunking, the recursive function first invokes a CRF chunker to obtain chunking hypotheses for the given sequence. For each hypothesis whose probability is high enough to have possibility of constituting the best derivation, the function calls itself with the sequence updated by the hypothesis. The parsing process is performed in a bottom up manner and this recursive process terminates if the whole sequence is chunked as a sentence.

To extract multiple chunking hypotheses from the CRF chunker, we use a branch-and-bound algorithm rather than the A* search algorithm, which is perhaps more commonly used in previous studies. We do not give pseudo code, but the basic idea is as follows. It first performs the forward Viterbi algorithm to obtain the best sequence, storing the upper bounds that are used for pruning in branch-and-bound. It then performs a branch-and-bound algorithm in a backward manner to retrieve possible candidate sequences whose probabilities are greater than the given threshold. Unlike A* search, this method is memory efficient because it is performed in a depth-first manner and does not require priority queues for keeping uncompleted hypotheses.

It is straightforward to introduce beam search in this search algorithm—we simply limit the number of hypotheses generated by the CRF chunker. We examine how the width of the beam affects the parsing performance in the experiments.

Symbol Unigrams	$s_{-2},\ s_{-1},\ s_0,\ s_{+1},\ s_{+2}$
Symbol Bigrams	$s_{-2}s_{-1},\ s_{-1}s_0,\ s_0s_{+1},\ s_{+1}s_{+2}$
Symbol Trigrams	$s_{-3}s_{-2}s_{-1},\ s_{-2}s_{-1}s_0,\ s_{-1}s_0s_{+1},\ s_0s_{+1}s_{+2},\ s_{+1}s_{+2}s_{+3}$
Head Unigrams	$h_{-2},\ h_{-1},\ h_0,\ h_{+1},\ h_{+2}$
Head Bigrams	$h_{-2}h_{-1},\ h_{-1}h_0,\ h_0h_{+1},\ h_{+1}h_{+2}$
Head Trigrams	$h_{-1}h_0h_{+1}$
Symbol & Daughters	s_0d_{01}, ... s_0d_{0m}
Symbol & Word/POS context	$s_0w_{j-1},\ s_0p_{j-1},\ s_0w_{k+1}\ ,\ s_0p_{k+1}$
Symbol & Words on the edges	$s_0w_j,\ s_0w_k$
Freshness	whether s_0 has been created in the level just below
PP-attachment	$h_{-1}h_0m_{02}$ (only when s_0 = PP)

Table 2: Feature templates used in the upper level chunking. s represents a non-terminal symbol. h represents a head percolated from the bottom for each symbol. d_{0i} is the ith daughter of s_0. w_j is the first word in the range covered by s_0. w_{j-1} is the word preceding w_j. w_k is the last word in the range covered by s_0. w_{k+1} is the word following w_k. p represents POS tags. m_{02} represents the head of the second daughter of s_0.

Word Unigram	$w_{-2}, w_{-1}, w_0, w_{+1}, w_{i+2}$
Word Bigram	$w_{-1}w_0, w_0w_{+1}, w_{-1}w_{+1}$
Prefix, Suffix	prefixes of w_0 suffixes of w_0 (up to length 10)
Character features	w_0 has a hyphen w_0 has a number w_0 has a capital letter w_0 is all capital
Normalized word	N(w_0)

Table 3: Feature templates used in the POS tagger. w represents a word and the subscript represents a relative position.

5 Part-of-Speech Tagging

We use the CRF model also for POS tagging. The CRF-based POS tagger is incorporated in the parser in exactly the same way as the other layers of chunking. In other words, the POS tagging process is treated like the bottom layer of chunking, so the parser considers multiple probabilistic hypotheses output by the tagger in the search algorithm described in the previous section.

5.1 Features

Table 3 shows the feature templates used in the POS tagger. Most of them are standard features commonly used in POS tagging for English. We used unigrams and bigrams of neighboring words, prefixes and suffixes of the current word, and some characteristics of the word. We also normalized the current word by lowering capital letters and converting all the numerals into '#', and used the normalized word as a feature.

6 Experiments

We ran parsing experiments using the Wall Street Journal corpus. Sections 2-21 were used as the training data. Section 22 was used as the development data, with which we tuned the feature set and parameters for learning and parsing. Section 23 was reserved for the final accuracy report.

The training data for the CRF chunkers were created by converting each parse tree in the training data into a list of chunking sequences like the ones presented in Figures 1 to 4. We trained three CRF models, i.e., the POS tagging model, the base chunking model, and the non-base chunking model. The training took about two days on a single CPU.

We used the *evalb* script provided by Sekine and Collins for evaluating the labeled recall/precision of the parser outputs[2]. All experiments were carried out on a server with 2.2 GHz AMD Opteron processors and 16GB memory.

6.1 Chunking Performance

First, we describe the accuracy of individual chunking processes. Table 4 shows the results for the ten most frequently occurring symbols on the development data. Noun phrases (NP) are the

[2]The script is available at http://nlp.cs.nyu.edu/evalb/. We used the parameter file "COLLINS.prm".

```
1:  procedure PARSESENTENCE(x)
2:      PARSE(x, 1, 0)
3:
4:  function PARSE(x, p, q)
5:      if x is chunked as a complete sentence
6:          return p
7:      H ← PERFORMCHUNKING(x, q/p)
8:      for h ∈ H in descending order of their
        probabilities do
9:          r ← p × h.probability
10:         if r > q then
11:             x' ← UPDATESEQUENCE(x, h)
12:             s ← PARSE(x', r, q)
13:             if s > q then
14:                 q ← s
15:     return q
16:
17: function PERFORMCHUNKING(x, t)
18:     perform chunking with a CRF chunker and
19:     return a set of chunking hypotheses whose
20:     probabilities are greater than t.
21:
22: function UPDATESEQUENCE(x, h)
23:     update sequence x according to chunking
24:     hypothesis h and return the updated
25:     sequence.
```

Figure 6: Searching for the best parse with a depth-first search algorithm. This pseudo-code illustrates how to find the highest probability parse, but in the real implementation, the function needs to keep track of chunking histories as well as probabilities.

most common symbol and consist of 55% of all phrases. The accuracy of noun phrases recognition was relatively high, but it may be useful to design special features for this particular type of phrase, considering the dominance of noun phrases in the corpus. Although not directly comparable, Sha and Pereira (2003) report almost the same level of accuracy (94.38%) on noun phrase recognition, using a much smaller training set. We attribute their superior performance mainly to the use of second-order features on state transitions. Table 4 also suggests that adverb phrases (ADVP) and adjective phrases (ADJP) are more difficult to recognize than other types of phrases, which coincides with the result reported in (Collins, 1999).

It should be noted that the performance reported in this table was evaluated using the gold standard sequences as the input to the CRF chunkers. In the real parsing process, the chunkers have to use the output from the previous (one level below) chunker, so the quality of the input is not as good as that used in this evaluation.

Symbol	# Samples	Recall	Prec.	F-score
NP	317,597	94.79	94.16	94.47
VP	76,281	91.46	91.98	91.72
PP	66,979	92.84	92.61	92.72
S	33,739	91.48	90.64	91.06
ADVP	21,686	84.25	85.86	85.05
ADJP	14,422	77.27	78.46	77.86
QP	14,308	89.43	91.16	90.28
SBAR	11,603	96.42	96.97	96.69
WHNP	8,827	95.54	97.50	96.51
PRT	3,391	95.72	90.52	93.05
:	:	:	:	:
all	579,253	92.63	92.62	92.63

Table 4: Chunking performance (section 22, all sentences).

Beam	Recall	Prec.	F-score	Time (sec)
1	86.72	87.83	87.27	16
2	88.50	88.85	88.67	41
3	88.69	89.08	88.88	61
4	88.72	89.13	88.92	92
5	88.73	89.14	88.93	119
10	88.68	89.19	88.93	179

Table 5: Beam width and parsing performance (section 22, all sentences).

6.2 Parsing Performance

Next, we present the actual parsing performance. The first set of experiments concerns the relationship between the width of beam and the parsing performance. Table 5 shows the results obtained on the development data. We varied the width of the beam from 1 to 10. The beam width of 1 corresponds to deterministic parsing. Somewhat unexpectedly, the parsing accuracy did not drop significantly even when we reduced the beam width to a very small number such as 2 or 3.

One of the interesting findings was that recall scores were consistently lower than precision scores throughout all experiments. A possible reason is that, since the score of a parse is defined as the product of all chunking probabilities, the parser could prefer a parse tree that consists of a small number of chunk layers. This may stem

from the history-based model's inability of properly trading off decisions made by different chunkers.

Overall, the parsing speed was very high. The deterministic version (beam width = 1) parsed 1700 sentences in 16 seconds, which means that the parser needed only 10 msec to parse one sentence. The parsing speed decreases as we increase the beam width.

The parser was also memory efficient. Thanks to L1 regularization, the training process did not result in many non-zero feature weights. The numbers of non-zero weight features were 58,505 (for the base chunker), 263,889 (for the non-base chunker), and 42,201 (for the POS tagger). The parser required only 14MB of memory to run.

There was little accuracy difference between the beam width of 4 and 5, so we adopted the beam width of 4 for the final accuracy report on the test data.

6.3 Comparison with Previous Work

Table 6 shows the performance of our parser on the test data and summarizes the results of previous work. Our parser achieved an f-score of 88.4 on the test data, which is comparable to the accuracy achieved by recent discriminative approaches such as Finkel et al. (2008) and Petrov & Klein (2008), but is not as high as the state-of-the-art accuracy achieved by the parsers that can incorporate global features such as Huang (2008) and Charniak & Johnson (2005). Our parser was more accurate than traditional history-based approaches such as Sagae & Lavie (2006) and Ratnaparkhi (1997), and was significantly better than previous cascaded chunking approaches such as Tsuruoka & Tsujii (2005) and Tjong Kim Sang (2001).

Although the comparison presented in the table is not perfectly fair because of the differences in hardware platforms, the results show that our parsing model is a promising addition to the parsing frameworks for building a fast and accurate parser.

7 Discussion

One of the obvious ways to improve the accuracy of our parser is to improve the accuracy of individual CRF models. As mentioned earlier, we were not able to use second-order features on state transitions, which would have been very useful, due to the problem of computational cost. Incremental feature selection methods such as grafting (Perkins et al., 2003) may help us to incorporate such higher-order features, but the problem of decreased efficiency of dynamic programming in the CRF would probably need to be addressed.

In this work, we treated the chunking problem as a sequence labeling problem by using the BIO representation for the chunks. However, semi-Markov conditional random fields (semi-CRFs) can directly handle the chunking problem by considering all possible combinations of subsequences of arbitrary length (Sarawagi and Cohen, 2004). Semi-CRFs allow one to use a richer set of features than CRFs, so the use of semi-CRFs in our parsing framework should lead to improved accuracy. Moreover, semi-CRFs would allow us to incorporate some useful restrictions in producing chunking hypotheses. For example, we could naturally incorporate the restriction that every chunk has to contain at least one symbol that has just been created in the previous level[3]. It is hard for the normal CRF model to incorporate such restrictions.

Introducing latent variables into the CRF model may be another promising approach. This is the main idea of Petrov and Klein (2008), which significantly improved parsing accuracy.

A totally different approach to improving the accuracy of our parser is to use the idea of "self-training" described in (McClosky et al., 2006). The basic idea is to create a larger set of training data by applying an accurate parser (e.g. reranking parser) to a large amount of raw text. We can then use the automatically created treebank as the additional training data for our parser. This approach suggests that accurate (but slow) parsers and fast (but not-so-accurate) parsers can actually help each other.

Also, since it is not difficult to extend our parser to produce N-best parsing hypotheses, one could build a fast reranking parser by using the parser as the base (hypotheses generating) parser.

8 Conclusion

Although the idea of treating full parsing as a series of chunking problems has a long history, there has not been a competitive parser based on this parsing framework. In this paper, we have demonstrated that the framework actually enables us to

[3]For example, the sequence VBD DT JJ in Figure 2 cannot be a chunk in the current level because it would have been already chunked in the previous level if it were.

	Recall	Precision	F-score	Time (min)
This work (deterministic)	86.3	87.5	86.9	0.5
This work (search, beam width = 4)	88.2	88.7	88.4	1.7
Huang (2008)			91.7	Unk
Finkel et al. (2008)	87.8	88.2	88.0	>250*
Petrov & Klein (2008)			88.3	3*
Sagae & Lavie (2006)	87.8	88.1	87.9	17
Charniak & Johnson (2005)	90.6	91.3	91.0	Unk
Tsuruoka & Tsujii (2005)	85.0	86.8	85.9	2
Collins (1999)	88.1	88.3	88.2	39**
Tjong Kim Sang (2001)	78.7	82.3	80.5	Unk
Charniak (2000)	89.6	89.5	89.5	23**
Ratnaparkhi (1997)	86.3	87.5	86.9	Unk

Table 6: Parsing performance on section 23 (all sentences). * estimated from the parsing time on the training data. ** reported in (Sagae and Lavie, 2006) where Pentium 4 3.2GHz was used to run the parsers.

build a competitive parser if we use CRF models for each level of chunking and a depth-first search algorithm to search for the highest probability parse.

Like other discriminative learning approaches, one of the advantages of our parser is its generality. The design of our parser is very generic, and the features used in our parser are not particularly specific to the Penn Treebank. We expect it to be straightforward to adapt the parser to other projective grammars and languages.

This parsing framework should be useful when one needs to process a large amount of text or when real time processing is required, in which the parsing speed is of top priority. In the deterministic setting, our parser only needed about 10 msec to parse a sentence.

Acknowledgments

This work described in this paper has been funded by the Biotechnology and Biological Sciences Research Council (BBSRC; BB/E004431/1) and the European BOOTStrep project (FP6 - 028099). The research team is hosted by the JISC/BBSRC/EPSRC sponsored National Centre for Text Mining.

References

Galen Andrew and Jianfeng Gao. 2007. Scalable training of L1-regularized log-linear models. In *Proceedings of ICML*, pages 33–40.

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

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of CoNLL-X*, pages 149–164.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of ACL*, pages 173–180.

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

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING 2004*, pages 282–288.

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

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08:HLT*, pages 959–967.

Jianfeng Gao, Galen Andrew, Mark Johnson, and Kristina Toutanova. 2007. A comparative study of parameter estimation methods for statistical natural language processing. In *Proceedings of ACL*, pages 824–831.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08:HLT*, pages 586–594.

Aravind K. Joshi and Phil Hopely. 1996. A parser from antiquity. *Natural Language Engineering*, 2(4):291–294.

Kevin Knight and Daniel Marcu. 2000. Statistics-based summarization - step one: Sentence compression. In *Proceedings of AAAI/IAAI*, pages 703–710.

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

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of HLT-NAACL*.

Yusuke Miyao, Rune Saetre, Kenji Sage, Takuya Matsuzaki, and Jun'ichi Tsujii. 2008. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of ACL-08:HLT*, pages 46–54.

Takashi Ninomiya, Takuya Matsuzaki, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2006. Extremely lexicalized models for accurate and fast HPSG parsing. In *Proceedings of EMNLP 2006*, pages 155–163.

Simon Perkins, Kevin Lacker, and James Theiler. 2003. Grafting: fast, incremental feature selection by gradient descent in function space. *The Journal of Machine Learning Research*, 3:1333–1356.

Slav Petrov and Dan Klein. 2008. Discriminative log-linear grammars with latent variables. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 1153–1160.

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

Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of COLING/ACL*, pages 691–698.

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

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

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. Chunk parsing revisited. In *Proceedings of IWPT*, pages 133–140.

Xiaofeng Yang, Jian Su, and Chew Lim Tan. 2006. Kernel-based pronoun resolution with structured syntactic features. In *Proceedings of COLING/ACL*, pages 41–48.