Chunk Parsing Revisited

Yoshimasa Tsuruoka[1,2] and Jun'ichi Tsujii[2,1]

[1] CREST, JST (Japan Science and Technology Agency)
Honcho 4-1-8, Kawaguchi-shi, Saitama 332-0012
[2] University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033
{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. Experimental results with the Penn Treebank corpus show that our chunk parser can give high-precision parsing outputs (88.8% precision and 81.0% recall) with extremely high speed (14 msec/sentence). We also show that extra search can improve the performance.

1 Introduction

Chunk parsing [1, 2] 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 covert 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 [1]. However, there is a very large gap between their performance and that of widely-used practical parsers [3, 4].

We show in this paper that the chunk parsing strategy is indeed appealing in that it can give considerably better performance than previously reported and that it enables us to build an extremely 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.

2 Chunk Parsing

For the overall strategy of chunk parsing, we follow the method proposed by Sang [1]. Figures 1 to 4 show an example of chunk parsing. In the first iteration, the chunker identifies two

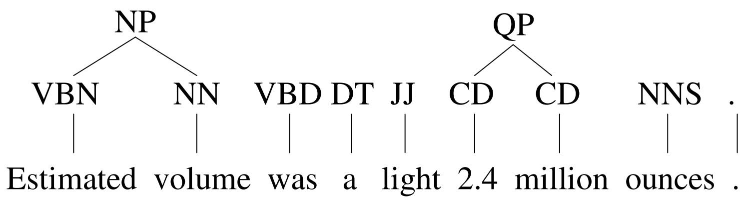

Fig. 1. Chunk parsing, the 1st iteration.

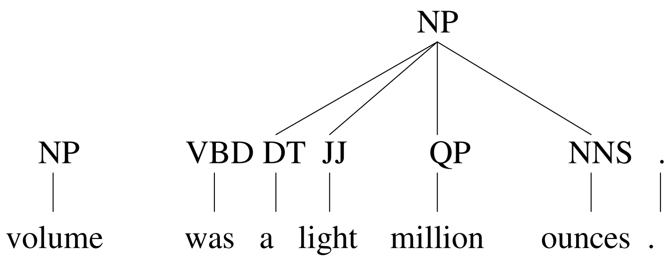

Fig. 2. Chunk parsing, the 2nd iteration.

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 [5]. In the second iteration, the chunker identifies (NP a light million ounces) and convert 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 (phrases). Sang used the IOB tagging method proposed by Ramshow[6] and memory-based learning for each level of chunking and achieved an f-score of 80.49% on the Penn Treebank corpus.

3 Chunking

The performance of chunk parsing heavily depends on the performance of each level of chunking. For the purpose of shallow parsing, the popular approach is to covert 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 [7], and Conditional Random Fields [8].

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 sub-sequences as phrase candidates and classify them with a machine learning

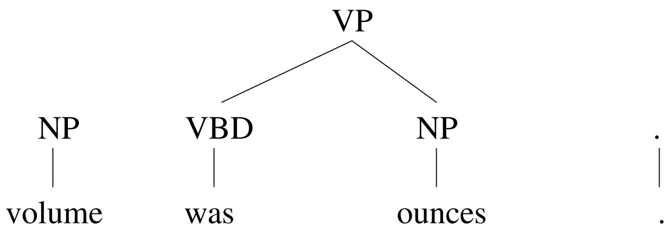

Fig. 3. Chunk parsing, the 3rd iteration.

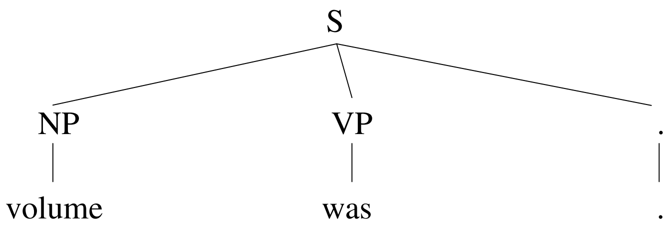

Fig. 4. Chunk parsing, the 4th iteration.

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 nonterminal symbols of the target) and the surrounding context, which are, in general, not available in sequential labeling approaches.

We should mention here that some of the sequence labeling approaches allows us to define arbitrary features on the target phrase. Semi-markov conditional random fields (Semi-CRFs) are one of such modeling methods [9]. Semi-CRFs could give better performance than the sliding-window approach because they can incorporate features on other phrase candidates

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

Table 1. Normalizing preterminals.

on the same level of chunking. However, they require additional computational resources for training and decoding, and the use of Semi-CRFs is future work.

The biggest disadvantage of the sliding window approach is, of course, the cost for training and testing. Since there are $n(n-1)$ 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 phrases in both training and testing. 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 [10] 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 testing, 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.

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 drastically reduces the sparseness problem.

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".

5 Filtering with the Naive Bayes classifier

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

To further reduce the cost required for training and testing, 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 naive Bayes classifier for each phrase type using the entire training data and used the following information as the features.

– The Right-Hand-Side (RHS) of the CFG rule
– The adjacent non-terminal symbol on the left.
– The adjacent non-terminal symbol on the right.

Thus, the probability for filtering is as follows:

$$P(y|c,l,r) = \frac{P(c,l,r|y)P(y)}{P(c,l,r)} \tag{1}$$

$$= \frac{P(c|y)P(l|y)P(r|y)P(y)}{\sum_y P(c|y)P(l|y)P(r|y)P(y)}, \tag{2}$$

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.

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 reduces the number of candidates with little positive samples that are 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 a maximum entropy classifier [11].

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.

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 probabilities 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 [12]. This modeling has one parameter to tune as in Gaussian prior modeling. The parameter is called *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 [13].

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 important. We use unigrams and bigrams of the neighboring heads. The RHS of the CFG rule is also useful. 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 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

In this paper, we also 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 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.

$$score = \prod_i P_i, \tag{3}$$

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

In each level of chunking, the chunker provides multiple possibilities of chunking together with their probabilities. The parser explores the best parse with the scoring scheme in a depth-first manner.

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 meta parameter for machine learning. The test set consists of section 23 and we report the performance of the parser on the set.

Threshold	Recall	Precision	F-score	Time (sec)
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.

Threshold	Recall	Precision	F-score	Time (sec)
0.5	75.64	90.79	82.53	41.9
0.4	79.63	90.14	84.56	77.6
0.3	82.64	89.43	85.90	85.9
0.2	84.69	88.58	86.55	109.1
0.1	84.79	86.71	85.74	124.0

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

We used the *evalb* script provided by Sekine and Collins for evaluating the labeled recall/precision of the parser outputs [3]. All the experiments were carried out on a server having a 2.6 GHz 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 tags. The trade-off between precision and recall can be controlled by changing the threshold for recognizing chunks. The first 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%, which is considerably better than the previously reported score (80.49%) of chunk parsing [1].

The parsing speed is extremely 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).

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

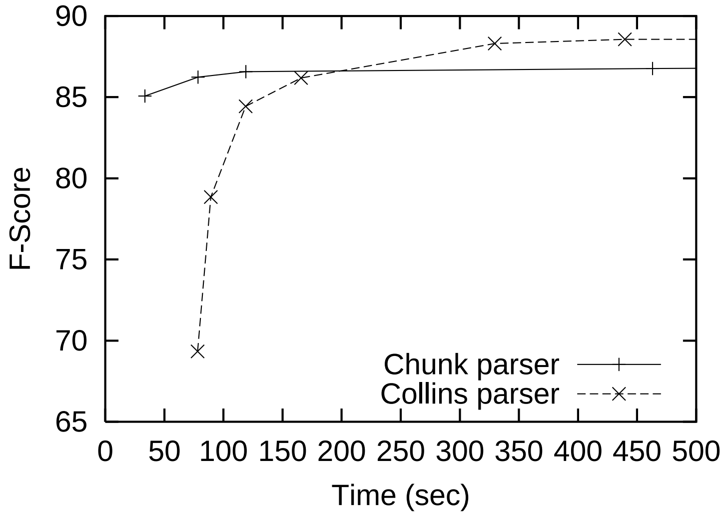

Fig. 5. Time vs F-score on section 23. The x-axis represents the time required to parse the entire section. The time required for making a hash table in Collins parser is not included.

Table 5 shows the performance achieved when we used the search algorithm described in Section 7.2. The search algorithm significantly boosted the precisions and recalls and achieved an f-score of 86.55% when the threshold was 0.2. It should be noted that we obtain no gain when we use a tight threshold. We need to consider phrases having low probability in order for the search to work.

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 [4] and our chunk parser in Figure 5. Collins parser allows the user to change the size of the beam in parsing. We used Model-2 because it gives better performance than Model-3 when the beam size is smaller than 1000. As for the chunk parser, we controlled the trade-off by changing the maximum number of nodes in the search. The threshold probability for chunk recognition was 0.2. Figure 5 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).

9 Conclusion

In this paper we show that chunk parsing can perform significantly better than previously reported even with a very simple set of features. Experimental results with the Penn Treebank corpus show that our chunk parser can give high-precision parsing outputs (88.8% precision and 81.0% recall) with extremely high speed (14 msec/sentence). We also show that searching can improve the performance and the f-score reaches 86.55%. Although there is still a gap between our f-score 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.

For future work, we plan to use additional feature sets for chunking. The chunk 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 [14–16], we expect that additional features will improve the performance of chunk parsing.

References

1. Tjong Kim Sang, E.: Transforming a chunker to a parser. In W. Daelemans, K. Sima'an, J.V., Zavrel, J., eds.: Computational Linguistics in the Netherlands 2000. Rodopi (2001) 177–188
2. Brants, T.: Cascaded markov models. In: Proceedings of EACL 1999. (1999)
3. Charniak, E.: A maximum-entropy-inspired parser. In: Proceedings of NAACL 2000. (2000) 132–139
4. Collins, M.: Head-Driven Statistical Models for Natural Language Parsing. PhD thesis, University of Pennsylvania (1999)
5. Magerman, D.M.: Statistical decision-tree models for parsing. In: Proceedings of ACL 1995. (1995) 276–283
6. Ramshaw, L., Marcus, M.: Text chunking using transformation-based learning. In Yarovsky, D., Church, K., eds.: Proceedings of the Third Workshop on Very Large Corpora, Somerset, New Jersey, Association for Computational Linguistics (1995) 82–94
7. Kudo, T., Matsumoto, Y.: Chunking with support vector machines. In: Proceedings of NAACL 2001. (2001)
8. Sha, F., Pereira, F.: Shallow parsing with conditional random fields. In: Proceedings of HLT-NAACL 2003. (2003)
9. Sarawagi, S., Cohen, W.W.: Semi-markov conditional random fields for information extraction. In: Proceedings of ICML 2004. (2004)
10. Ratnaparkhi, A.: A maximum entropy model for part-of-speech tagging. In Brill, E., Church, K., eds.: Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Somerset, New Jersey (1996) 133–142
11. Berger, A.L., Pietra, S.A.D., Pietra, V.J.D.: A maximum entropy approach to natural language processing. Computational Linguistics **22** (1996) 39–71
12. Kazama, J., Tsujii, J.: Evaluation and extension of maximum entropy models with inequality constraints. In: Proceedings of EMNLP 2003. (2003)
13. Benson, S.J., Moré, J.: A limited-memory variable-metric algorithm for bound-constrained minimization. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory (2001) ANL/MCS-P909-0901.
14. Bod, R.: Data oriented parsing. In: Proceedings of COLING 1992. (1992)
15. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: Proceedings of ACL 2002. (2002)
16. Kudo, T., Suzuki, J., Isozaki, H.: Boosting-based parse reranking with subtree features. In: Proceedings of ACL 2005. (2005)