Transforming a Chunker to a Parser

Erik F. Tjong Kim Sang

CNTS - Language Technology Group
University of Antwerp
erikt@uia.ua.ac.be

Abstract

Ever since the landmark paper Ramshaw and Marcus (1995), machine learning systems have been used successfully for identifying base phrases (chunks), the bottom constituents of a parse tree. We expand a state-of-the-art chunking algorithm to a bottom-up parser by recursively applying the chunker to its own output. After testing different training configurations we obtain a reasonable parser which is tested against a standard data set. Its performance falls behind that of current state-of-the-art parsers. We give some suggestions for modifications of the parser which may lead to future performance improvements.

1 Introduction

Ramshaw and Marcus (1995) have proposed approaching base phrase identification by regarding it as a tagging task. Phrases in a text can be represented by word-related phrase chunk tags, like I (inside chunk) and O (outside chunk). The advantage of this approach is that arbitrary simple machine learning algorithms, for example classifiers, can be used for performing this task. This paper has initiated a lot of follow-up research, both regarding base noun phrase identification and finding arbitrary base chunks.

A group of chunkers can build a complete parse tree if each of them identifies syntactic chunks at a different level of the tree. The first chunker finds base chunks and send these to the second chunker which identifies syntactic constituents one level above the base level. This chunker sends its output to another one which identifies constituents at the next level and so on. The chunkers can be implemented with any classification algorithm which enables researchers from different segments of the machine learning field to build a parser.

The idea of using chunkers in a parser is not new. Ejerhed and Church (1983) describe a grammar for Swedish which includes noun phrase chunk rules. Abney (1991) built a chunk parser which first finds base chunks and then attaches them with a separate attachment process. Brants (1999) used a cascade of Markov model chunkers for obtaining parsing results for the German NEGRA corpus.

In this paper, we expand a state-of-the art chunking algorithm to an algorithm for full parsing. Our approach is most similar to Brants (1999) : we use a chunker for retrieving all parse tree levels. However, we apply the chunk parser to a standard data set for English in order to allow a comparison with earlier parser results. We describe the chunker-parser transformation process in detail and examine different training configurations of the chunk parser. After this we test the parser and compare its performance with a state-of-the-art statistical parser.

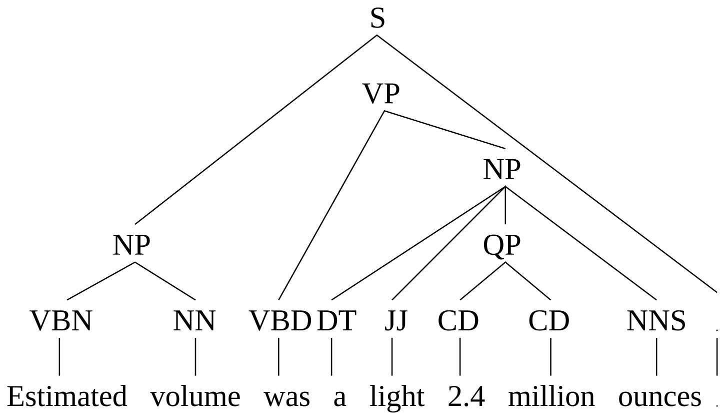

Figure 1: Example parse tree from the Penn Treebank. The bottom row contains the words (*Estimated...*), the row above that one contains the part-of-speech tags (VBN...) and the next the chunk tags (NP QP).

2 Chunking

Identifying the phrases at the bottom part of the tree is called chunking. For example, the tree in figure 1 contains two base phrases, a noun phrase and a quantifier phrase:

> (NP Estimated volume) was
> a light (QP 2.4 million) ounces .

The first report on applying statistical methods to this task dates back to 1988 (Church 1988). The most influential work in this area is Ramshaw and Marcus (1995). They showed that this task can be approached as a tagging task by replacing the common bracket notation for chunks with a tagging representation:

> Estimated/B-NP volume/I-NP was/O
> a/O light/O 2.4/B-QP million/I-QP ounces/O ./O

Their tagging scheme contains three types of tags: B-XX for the first word of a chunk of type XX, I-XX for non-initial words in such chunks and O for words outside of any chunk.[1] The tagging representation has an advantages over the bracket representation: while the latter may suffer from bracket-pairing problems, the tagging representation requires almost no consistency checks. Furthermore it allows existing tagging technology to be used for deriving chunks in text.

[1] The original Ramshaw and Marcus (1995) chunk format was slightly different from the one shown here. It included an extra punctuation chunk type (P) and the tags did not contain hyphens.

	IOB1	IOB2	IOE1	IOE2	O+C	
Estimated	I-NP	B-NP	I-NP	I-NP	[-NP	O
volume	I-NP	I-NP	I-NP	E-NP	O	]-NP
was	O	O	O	O	O	O
a	O	O	O	O	O	O
light	O	O	O	O	O	O
2.4	I-QP	B-QP	I-QP	I-QP	[-QP	O
million	I-QP	I-QP	I-QP	E-QP	O	]-QP
ounces	O	O	O	O	O	O
.	O	O	O	O	O	O

Table 1: Five representations of the chunk structure of the sentence *Estimated volume was a light 2.4 million ounces* . IOB1 and IOE1 only use a B-XX or an E-XX tag at the boundaries of adjacent chunks of the same type (not present in this sentence). IOB2 and IOE2 use B-XX and E-XX tags at boundaries of all chunks. O+C is the bracket representation.

Recent work has shown that with a good method for repairing bracket problems, a noun phrase chunking system using the bracket representation performs better than one that uses the tagging representation (Muñoz, Punyakanok, Roth and Zimak 1999). Even more promising is the approach that combines the output of chunkers that use different data representations (Tjong Kim Sang 2000a). The idea is build five chunking systems, one which uses the bracket representation and four which use variants of the tagging representation (see table 1). Because of the different formats of the data, the five systems will make different errors. After converting their output to the bracket representation, one can extract a new chunk segmentation by only accepting brackets which have been predicted by the majority of the systems. The new chunk segmentation proves to be better than any generated by the individual systems (Tjong Kim Sang 2000a).

We have used the combination of representations chunking method of Tjong Kim Sang (2000b). This means training five classifiers with each of the data representations shown in table 1. Each of the classifiers performs four passes over the data. First they determine the chunk boundaries regardless of the type (two passes for the bracket representation). After this the four systems that use the tagging representation use the chunk structure found as extra information for producing an improved segmentation. Next all data is converted to the bracket representation and each system performs two passes for determining the types of the open and close brackets, respectively. This process with four passes over the data was chosen after a comparative chunking study (Tjong Kim Sang 2000b). An example of the features used in the four passes can be found in figure 2. When the output of the five systems is available, the five open bracket and five close bracket data streams are combined with majority voting. Finally, the resulting open and close brackets are made consistent by throwing away all brackets that cannot be matched with an adjacent bracket of the same chunk type.

Est JJ vol NN was VBD a DT lig JJ 2.4 CD mil CD oun NNS . .
vol NN I was VBD O a DT O lig JJ 2.4 CD B mil CD I oun NNS O

Figure 2: Features used for classifying *light* in the sentence *Estimated volume was a light 2.4 million ounces* . Word length has been limited three characters to make the lines fit in this figure. The first row contains the features used in the first, third and the fourth pass: the word and its part-of-speech (POS) tag with the four previous and four next words with their POS tags. The second row shows the features used in the second pass of the system that worked with the IOB2 tagging representation: the word, its POS tag and the three previous and three next words with their POS tag and the chunk tag found in pass one.

Various machine learning methods can be used for the basic task of determining the most appropriate chunk tag sequences for a text. We use the memory-based learning algorithm IB1-IG which is part of TiMBL package (Daelemans, Zavrel, van der Sloot and van den Bosch 1999). In memory-based learning the training data is stored and a new item is classified by the most frequent classification among training items which are closest to this new item. Data items are represented as sets of feature-value pairs. In IB1-IG each feature receives a weight which is based on the amount of information which it provides for computing the classification of the items in the training data. These feature weights are used for computing the distance between a pair of data items (Daelemans et al. 1999). IB1-IG has been used successfully on a large variety of natural language processing tasks.

The overall results of the chunking process are measured with labeled precision and recall, respectively the percentage of detected phrases that are correct and the percentage of phrases in the data that were found by the parser. We have combined these two in the $F_{\beta=1}$ rate which is equal to (2*precision*recall)/(precision+recall).

3 Parsing

In this paper we will perform parsing, generating syntactic analyses for sentences, with a chunking algorithm. Most parsing work describes a process for finding base chunks which are combined by another process to trees (for example Ejerhed and Church (1983) and Abney (1991)). However, like Brants (1999), we want to build a parser in which constituents et all levels of the parse tree are produced by a chunking algorithm. This is a natural follow-up of our earlier work (Tjong Kim Sang 2000a, Tjong Kim Sang 2000b) in which we build a noun phrase chunker, a noun phrase parser and a general chunker. Our general parser performs the following actions:

w	Estimated	volume	was	a	light	2.4	million	ounces	.
p	VBN	NN	VBD	DT	JJ	CD	CD	NNS	.
0	(NP	NP)				(QP	QP)		
0w		volume	was	a	light		million	ounces	.
0p		NP	VBD	DT	JJ		QP	NNS	.
1				(NP				NP)	
1w		volume	was					ounces	.
1p		NP	VBD					NP	.
2			(VP					VP)	
2w		volume	was						.
2p		NP	VP						.
3		(S							S)

Figure 3: An example of the parsing process: first the base chunks are identified (row 0). The chunks are compressed and replaced by their heads (0w) and labels (0p). Then the next level of phrases is identified (1) and compressed to head words (1w) and labels (1p). This is followed by two more levels of phrase identification (2 and 3) and another compression (2w and 2p).

1. use a tagger for finding a part-of-speech tag for each word,
2. use a chunker for identifying base phrases,
3. replace all identified phrases with their head and their label,
4. find ‘base’ phrases in the new data stream
5. if the previous step discovered new phrases then repeat steps 3-5.

In this set-up only step 4 requires a training phase. The other actions are either fixed (3 and 5) or depend on parser-external processes (1 and 2). Ideally steps 2 and 4 would be the same but unfortunately we have discovered that the combination of representations chunking approach does not work well for non-base phrases. The systems which use the tagging representation perform much worse than the one that uses the bracket representation. The reason for this is that, based on a limited context, it is hard to determine if a word should be included in a chunk at level *n* or at level *n+1*. With the bracket representation, incorrect brackets will be generated but these will be eliminated in the bracket combination process. However, the systems that use the tagging representation produce incorrect B and I tags which cause errors in their output. A combination of the five systems performs worse than the system that uses the bracket representation and therefore we have decided to use only the latter for step 4 of the chunk parser.

In order to be able to compare our results with earlier work like Collins (1999), we have used the Maximum Entropy tagger described in Ratnaparkhi (1996) for assigning part-of-speech tags to our data. The words in the data have been combined to base chunks by the chunking approach described in the previous section. After finding the base phrases, each of them will be replaced by their head (the

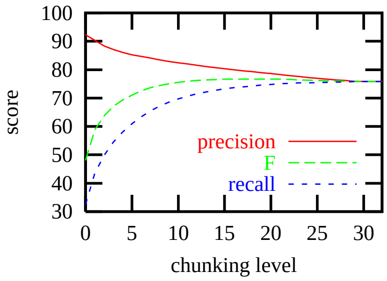

Figure 4: Precision, recall and $F_{\beta=1}$ rates for one run of the parser applied to the parameter tuning data set. The precision decreases when the chunking level increases, the recall increases at higher levels and the F rate increases until about level 19 and drops slightly afterwards (not noticeable in this graph).

most important word in the phrase) and their type, conform figure 3. The head word of each phrase is generated by a list of rules put forward by Magerman (1995) and modified by Collins (1999). [2] It chooses a head word of a phrase based on the type of the phrase, the types/POS tags of the phrases/words immediately under the phrase and the order of the latter. The approach of compressing identified phrases provides us with a standard data format. At all levels of the bottom-up process, sentences are represented by a sequence of lexical items (usually words) with associated syntactic tags (POS tags or phrase type tags).

The parsing process generates a unique tree for each sentence. The tree can be interpreted as a set of phrases. The output of the parser are evaluated with the same measures as the chunking results: labeled precision, labeled recall and $F_{\beta=1}$ rate, all applied to phrases. Additionally we have used the crossing brackets measure (CB) for indicating the average number of identified phrases per sentence that cross phrases in the original data.

4 Results

We have used two segment pairs of the Wall Street Journal (WSJ) part of the Penn Treebank (Marcus, Santorini and Marcinkiewicz 1993) for training and testing. First we used sections 15-18 (8,936 sentences) as training material and section 20 (2012 sentences) as test material for determining the optimal parameters of the parser (tuning data). The best configuration for that data was used for processing section 23 (2416 sentences) after training with sections 02-21 (39,832 sentences). The results for that data set have been used for a comparison with performance of other parsers.

In our first experiment we tested the influence of varying context size and

[2] Available on http://www.research.att.com/˜mcollins/papers/heads

	k=1	k=3
c=1	74.13 (18)	70.20 (13)
c=2	77.17 (19)	75.13 (15)
c=3	77.05 (16)	76.02 (16)
c=4	76.33 (15)	75.33 (14)

Table 2: $F_{\beta=1}$ rates for four context sizes combined with two sizes of nearest neighborhood. The parser was tested on the tuning data. The numbers between brackets are the optimal levels at which processing was stopped. We obtained the best $F_{\beta=1}$ rate for context size 2 combined with nearest neighborhood size 1.

neighborhood size in step 4 of the parser, the non-base chunking process. The context size is the number of words and syntactic tags that are used as features, for example context size 2 means that the classification of a word is determined while using the word, the two previous words, the next two words and the syntactic tags of all these words. The size of the neighborhood size determines the number of nearest neighbor regions that the memory-based learner uses to determine the class of the current data item. We have performed experiments with the tuning data with four different symmetric context sizes (1, 2, 3 and 4) and two neighborhood sizes (1 and 3).

Our parsing process works bottom-up, level by level. At each level it finds new phrases and therefore the number of detected phrases is expected to increase which means that the recall rate increases. Unfortunately, since we use one level of training data at a time (like Brants (1999)), the amount of available training data decreases at higher levels and therefore the precision of the recognition process decreases at higher levels (figure 4). Our goal is to optimize the $F_{\beta=1}$ rate of the results. In our experiments with the tuning data set, we have found that this rate reaches an optimum after about 15 levels of phrase recognition. At that point the parser is still finding new phrases but the improved recall rate cannot compensate for the drop in precision rate. For this reason we have decided to stop the cascaded phrase recognition processes in the tuning experiments when the F rate was optimal. We have added a post-processing stage which adds a top S node to trees which do not contain such a node. The results of the eight experiments, F rates and halting levels, can be found in table 2. The configuration with context size 2 and nearest neighborhood size 1 performed best. The best context size found here is the same as used by Ratnaparkhi (1998).

In the next series of experiments, we tested the influence of training data segmentation on parsing performance. In our previous experiments, we only used training data from one level at a time. For example, during identification of phrases at level 1, the ones immediately above the base chunks, we only used training data from level 1. This causes problems at higher levels, for which there are fewer examples available. Ideally, we should be able to use one body of training data for all levels. We have performed two extra experiments with context size 2 and

	$F_{\beta=1}$
using current training level only	77.17 (19)
using current, previous and next	77.13 (17)
using all training data	67.69 (20)
disregarding open bracket types	72.33 (24)
disregarding close bracket types	76.06 (29)

Table 3: The performance of five training variants. In these experiments we used the same context size (2) and nearest neighborhood size (1) but varied the amount of training material used at each chunk level or disregarded the types of the open brackets or close brackets during the bracket combination process. We obtained the best performance when using training data from the current chunking level.

nearest neighborhood size 1: one using all training data apart from the base chunk level (levels 1-31) and another which uses the previous, current and next level. The results can be found in table 3. Neither of the two set-ups performed better than the configuration which uses one level of training data at a time.

Our bracket combination algorithm is very strict: it only uses adjacent brackets if they have the same type and appear next to each other in the correct order (first open and then close) without intervening brackets. Chunkers which use this approach obtain high precision rates and low recall rates because they generate phrases which are likely to be correct and disregard all other phrases (Tjong Kim Sang 2000a). We have tested two alternative approaches: one which uses the type of the close brackets and disregards the type of the open brackets and another which uses the type of the open brackets and ignores close bracket types. The second approach performed better than the first but neither of them improved on the standard method we used for bracket combination (see table 3).

We have applied the best training configuration (context size 2, neighborhood size 1, train with current level and use 19 iterations plus post-processing stage) to the large data set. After training we obtained an $F_{\beta=1}$ rate of 80.49 on arbitrary sentences of the test data (precision 82.34%, recall 78.72% and crossing bracket rate 1.69). Figure 5 contains an overview of the progression of precision, recall and $F_{\beta=1}$ rate measured at different levels. The performance of the chunk parser is reasonable but it does worse than state-of-the-art statistical parsing systems (Collins 1999, Charniak 2000) which achieve $F_{\beta=1}$ rates close to 90 and CB rates under 1.10.[3]

[3] Obviously the parser has problems with deeply embedded sentences, because of the maximum parsing depth (19). However, its performance on short sentence (10 words or shorter) is not perfect either ($F_{\beta=1}$ = 89.5). We have not spotted systematic errors in the parsing output yet.

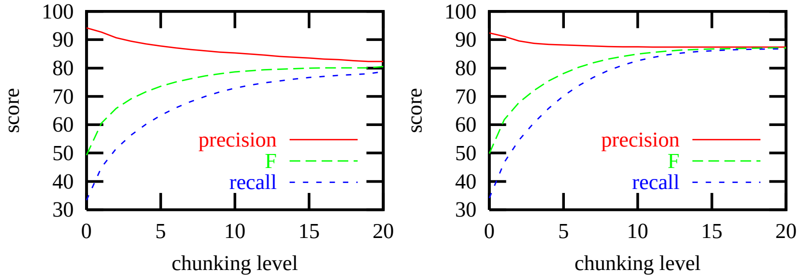

Figure 5: Per-level precision, recall and $F_{\beta=1}$ rates for sentences shorter than 100 words obtained by our chunk parser applied to section 23 of the Penn Treebank (left) compared with the parser of Collins (1999) (right). The parsers have generated 20 levels of chunk tags (0-19). An extra post-processing step (20) has added top S nodes to incomplete parse trees. The precision of the chunk parser drops faster and the recall grows slower than that of the Collins parser. The final F rates are respectively 80.49 and 87.08.

5 Related work

A lot of work has been done on natural language parsing. In this section we mention some of the work which uses machine learning methods for building parsers rather than hand-crafted grammars. Black et.al. (1992) introduce history-based grammars: a statistical parsing model which obtains probabilistic clues of the optimal parses from training data. It is hard to compare their performance with other work because they used different test data and different evaluation methods. Magerman (1995) built SPATTER, a statistical parser which uses decision trees. It obtained an F rate of 86.0 on sentences shorter than 40 words of section 00 of the Penn Treebank.

The parsing method used by Ratnaparkhi (1998) is closely related to our approach. He uses a classification algorithm, maximum entropy models, for building parse trees in a bottom-up, left-to-right fashion. The parser performs better than the previous two and obtains F=86.9 on Penn Treebank section 23. Collins (1999) describes different statistical parsing models. They use complex statistical predicates for deriving optimal parse trees. His best model obtains F=88.2 on section 23. Charniak (2000) combines earlier work by Ratnaparkhi, Collins and himself and creates a parser which obtains an F rate of 89.5 on section 23 of the Penn Treebank. Finally Bod (2000) mentions a parser which performs slightly better by using a big database of stored subtrees of limited depth.

Regarding parsing with chunking techniques, three more papers need to be mentioned. Ejerhed and Church (1983) approach parsing of Swedish by starting with identifying noun phrase chunks. Their approach requires a partial order the non-terminal symbols which limits the format of the trees that can be produced. Abney (1991) describes a parser which starts with identifying base phrases which are combined to a complete tree by an attachment process. Brants (1999) built a bottom-up parser which identifies chunks in German text with Markov models. An interesting feature of his parser is that it is capable of correcting errors made at earlier stages.

6 Discussion

The parsing method described here leaves open many options for future work. An important difference between statistical parsers and our approach is the information that is available to the parser. For example, our chunk parser does not have access to the internal structure of a chunk apart from its type and its head word. If we compare it performance with a study of Charniak (1997) , we note that our parser performs better than a standard probabilistic context-free grammar ($F_{\beta=1}$=73.7) but worse than than the Minimal extension of this grammar ($F_{\beta=1}$=82.8). However, even this Minimal extension is provided with more information than our parser: the rule probabilities depend on the head and the type of the phrase and the type of the parent. The bottom-up parser described in this paper does not have access to information about the parents. The availability of this information would probably improve the parser's performance although it would also complicate the chunk identification processes.

The parser would also benefit from using a less greedy method of bracket prediction which would enable it to backtrack from earlier phrase structures choices at later processing levels. At this moment, it attempts to find the best bracket structure at each level and sticks with that. This is different from most other parsers, like for example the one of Ratnaparkhi (1998) which is capable of storing up to 20 intermediate trees and examines structural extensions of all of them.

Another opportunity for performance enhancement lies in using a different bracket estimation algorithm. At this moment, we feel that we do not have used the best method for finding chunks at the non-base level. We have achieved a reasonable improvement by using combinations of classifiers for chunking rather than a single classifier (Tjong Kim Sang 2000a). Something similar should be possible while identifying higher level chunks, possibly by using a variety of machine learning algorithms. It would also be interesting to examine the process which combines open with close brackets. At this moment combinations are made without regarding more than two brackets. A combination method which examines the complete sentence before creating a bracket pair, like the one of Muñoz, Punyakanok, Roth and Zimak (1999), might improve performance.

Extra improvements of this parser will undoubtfully make it slower and increase its memory requirements. The present process already requires a lot of computational resources: close to 100 megabytes of internal memory for generating base chunks with the small training data at a processing speed of more than a second per word. This is orders of magnitude slower than for example Ratnaparkhi's parser which in 1998 achieved a processing speed of 0.14 seconds per sentence which achieving a higher performance rate (Ratnaparkhi 1998). It is doubtful whether extension of the work reported on here is worth the trouble.

7 Concluding remarks

We have presented a method for transforming a chunker to a full parser. It consists of using a cascade of phrase recognition algorithms for building parse trees in a bottom-up fashion. The advantage of this approach is that it enables building a parser by using the same simple machine learning algorithms (classifiers) which have successfully been used for base phrase recognition. We have tested this approach by converting a chunking algorithm to a parsing algorithm and evaluated the resulting chunk parser on a standard data set. The parser performed reasonably (F=80) but it did not reach the performances of the state-of-art statistical parsers (F=90).

References

Abney, S.(1991), Parsing by chunks, *Principle-Based Parsing*, Kluwer Academic Publishers.

Black, E., Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R. and Roukos, S.(1992), Towards history-based grammars: Using richer models for probabilistic parsing, *Proceedings DARPA Speech and Natural Language Workshop*, Morgan Kaufmann.

Bod, R.(2000), Parsing with the shortest derivation, *Proceedings of COLING 2000*, Saarbruecken, Germany.

Brants, T.(1999), Cascaded markov models, *Proceedings of EACL'99*, Bergen, Norway.

Charniak, E.(1997), Statistical parsing with a context-free grammar and word statistics, *Fourteenth National Conference on Artificial Intelligence*, MIT Press.

Charniak, E.(2000), A maximum-entropy-inspired parser, *Proceedings of the ANLP-NAACL 2000*, Seattle, WA, USA. Morgan Kaufman Publishers.

Church, K. W.(1988), A stochastic parts program and noun phrase parser for unrestricted text, *Second Conference on Applied Natural Language Processing*, Austin, Texas.

Collins, M.(1999), *Head-Driven Statistical Models for Natural Language Processing*, PhD thesis, University of Pennsylvania.

Daelemans, W., Zavrel, J., van der Sloot, K. and van den Bosch, A.(1999), *TiMBL: Tilburg Memory Based Learner, version 2.0, Reference Guide*, ILK Technical Report 99-01. http://ilk.kub.nl/.

Ejerhed, E. and Church, K. W.(1983), Finite state parsing, *Papers from the Seventh Scandinavian Conference of Linguistics*, University of Helsinki, Finland.

Magerman, D. M.(1995), Statistical decision-tree models for parsing, *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL'95)*, Cambridge, MA, USA.

Marcus, M. P., Santorini, B. and Marcinkiewicz, M. A.(1993), Building a large annotated corpus of english: the penn treebank, *Computational Linguistics*.

Muñoz, M., Punyakanok, V., Roth, D. and Zimak, D.(1999), A learning approach to shallow parsing, *Proceedings of EMNLP-WVLC'99*, Association for Computational Linguistics.

Ramshaw, L. A. and Marcus, M. P.(1995), Text chunking using transformation-based learning, *Proceedings of the Third ACL Workshop on Very Large Corpora*, Cambridge, MA, USA.

Ratnaparkhi, A.(1996), A maximum entropy model for part-of-speech tagging, *Proceedings of EMNLP-1*, University of Pennsylvania, PA, USA.

Ratnaparkhi, A.(1998), *Maximum Entropy Models for Natural Language Ambiguity Resolution*, PhD thesis Computer and Information Science, University of Pennsylvania.

Tjong Kim Sang, E. F.(2000a), Noun phrase recognition by system combination, *Proceedings of the ANLP-NAACL 2000*, Seattle, Washington, USA. Morgan Kaufman Publishers, pp. 50–55.

Tjong Kim Sang, E. F.(2000b), Text chunking by system combination, *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, pp. 151–153.