

Rapid Parser Development: A Machine Learning Approach for Korean

Ulf Hermjakob

USC Information Sciences Institute
4676 Admiralty Way #1000 · Marina del Rey, CA 90292 · USA
ulf@cs.utexas.edu

Abstract

This paper demonstrates that machine learning is a suitable approach for rapid parser development. From 1000 newly treebanked Korean sentences we generate a deterministic shift-reduce parser. The quality of the treebank, particularly crucial given its small size, is supported by a consistency checker.

1 Introduction

Given the enormous complexity of natural language, parsing is hard enough as it is, but often unforeseen events like the crises in Bosnia or East-Timor create a sudden demand for parsers and machine translation systems for languages that have not benefited from major attention of the computational linguistics community up to that point.

Good machine translation relies strongly on the context of the words to be translated, a context that often goes well beyond neighboring surface words. Often basic relationships, like that between a verb and its direct object, provide crucial support for translation. Such relationships are usually provided by parsers.

The NLP resources for a language of sudden international interest are typically quite limited. There is probably a dictionary, but most likely no treebank. Maybe basic tools for morphological analysis, but probably no semantic ontology.

This paper reports on the rapid development of a parser based on very limited resources. We show that by building a small treebank of only a thousand sentences, we could develop a good basic parser using machine learning within only three months. For the language we chose, Korean, a number of research groups have been working on parsing and/or machine translation in recent years (Yoon, 1997; Seo, 1998; Lee, 1997), but advanced resources have not been made publicly available, and we have not used any, thereby so-to-speak at least simulating a low density language scenario.

2 Korean

Like Japanese, Korean is a head-final agglutinative language. It is written in a phonetic alphabet called

hangul, in which each two-byte character represents one syllable. While our parser operates on the original Korean *hangul*, this paper presents examples in a romanized transcription. In sentence (1) for example, the verb is preceded by a number of so-called *eojeols* (equivalent to *bunsetsus* in Japanese) like “*chaeg-eul*”, which are typically composed of a content part (“*chaeg*” = book) and a postposition, which often corresponds to a preposition in English, but is also used as a marker of topic, subject or object (“*eul*”).

나는 어제 그 책을 샀다.
Na-neun eo-je geu chaeg-eul sass-da. (1)
I_{TOPIC} yesterday this book_{OBJ} bought.
I bought this book yesterday.

Our parser produces a tree describing the structure of a given sentence, including syntactic and semantic roles, as well as additional information such as tense. For example, the parse tree for sentence (1) is shown below:

```
[1] na-neun eo-je geu chaeg-eul sass-da. [S]
    (SUBJ) [2] na-neun [NP]
        (HEAD) [3] na [REG-NOUN]
        (PARTICLE) [4] neun [DUPLICATE-PRT]
    (TIME) [5] eo-je [REG-ADVERB]
        (HEAD) [6] eo-je [REG-ADVERB]
    (OBJ) [7] geu chaeg-eul [NP]
        (MOD) [8] geu [DEMONSTR-ADNOMINAL]
            (HEAD) [9] geu [DEMONSTR-ADNOMINAL]
        (HEAD) [10] chaeg-eul [NP]
            (HEAD) [11] chaeg [REG-NOUN]
            (PARTICLE) [12] eul [OBJ-CASE-PRT]
    (HEAD) [13] sass-da. [VERB; PAST-TENSE]
        (HEAD) [14] sa [VERB-STEM]
        (SUFFIX) [15] eoss [INTERMED-SUF-VERB]
        (SUFFIX) [16] da [CONNECTIVE-SUF-VERB]
        (DUMMY) [17] . [PERIOD]
```

Figure 1: Parse tree for sentence 1 (simplified)

For preprocessing, we use a segmenter and morphological analyzer, KMA, and a tagger, KTAG, both provided by the research group of Prof. Rim of

Korea University. KMA, which comes with a built-in Korean lexicon, segments Korean text into *eojeols* and provides a set of possible sub-segmentations and morphological analyses. KTAG then tries to select the most likely such interpretation. Our parser is initialized with the result of KMA, preserving all interpretations, but marking KTAG’s choice as the top alternative.

3 Treebanking Effort

The additional resources used to train and test a parser for Korean, which we will describe in more detail in the next section, were (1) a 1187 sentence treebank, (2) a set of 133 context features, and (3) background knowledge in form of an ‘*is-a*’ ontology with about 1000 entries. These resources were built by a team consisting of the principal researcher and two graduate students, each contributing about 3 months.

3.1 Treebank

The treebank sentences are taken from the Korean newspaper Chosun, two-thirds from 1994 and the remainder from 1999. Sentences represent continuous articles with no sentences skipped for length or any other reason. The average sentence length is 21.0 words.

3.2 Feature Set

The feature set describes the context of a partially parsed state, including syntactic features like the part of speech of the constituent at the front/top of the input list (as sketched in figure 2) or whether the second constituent on the parse stack ends in a comma, as well as semantic features like whether or not a constituent is a time expression or contains a location particle. The feature set can accommodate any type of feature as long as it is computable, and can thus easily integrate different types of background knowledge.

3.3 Background Knowledge

The features are supported by background knowledge in the form of an ontology, which for example has a time-particle concept with nine sub-concepts (accounting for 9 of the 1000 entries mentioned above). Most of the background knowledge groups concepts like particles, suffixes, units (e.g. for lengths or currencies), temporal adverbs – semantic classes that are not covered by part of speech information of the lexicon, yet provide valuable clues for parsing.

3.4 Time Effort

The first graduate student, a native Korean and linguistics major, hired for 11 weeks, spent about 2 weeks getting trained, 6 weeks on building two-thirds of the treebank, 2 weeks providing most background knowledge entries and 1 week helping to

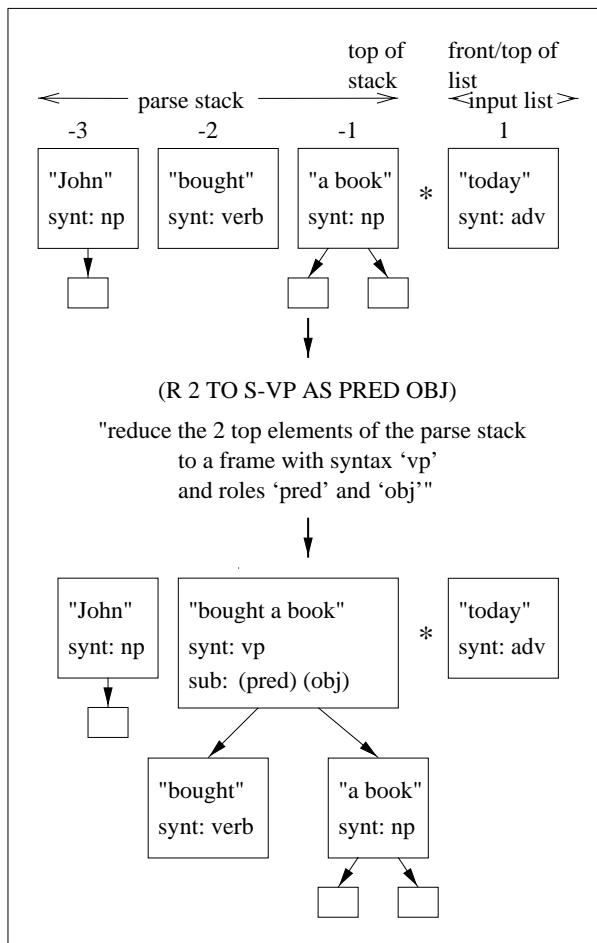


Figure 2: A typical parse action (simplified).

Boxes represent *frames*. The asterisk (*) represents the current parse position. Optionally, parse actions can have additional arguments, like target syntactic or semantic classes to overwrite any default. Elements on the input list are identified by positive integers, elements on the parse stack by negative integers. The feature ‘Synt of -1’ for example refers to the (main) syntactic category of the top stack element. Before the reduce operation, the feature ‘Synt of -1’ would evaluate to *np* (for “a book”), after the operation to *vp* (for “bought a book”). The input list is initialized with the morphologically analyzed words, possibly still ambiguous. After a sequence of shift (from input list to parse stack) and reduce (on the parse stack) operations, the parser eventually ends up with a single element on the parse stack, which is then returned as the parse tree.

identify useful features. The other graduate student, a native Korean and computer science major, installed Korean tools including a terminal for hangul and the above mentioned KMA and KTAG, wrote a number of scripts tying all tools together, made some tool improvements, built one-third of the treebank

and also contributed to the feature set. The principal researcher, who does not speak Korean, contributed about 3 person months, coordinating the project, training the graduate students, writing treebank consistency checking rules (see section 6), making extensions to the tree-to-parse-action-sequence module (see section 4.1) and contributing to the background knowledge and feature set.

4 Learning to Parse

We base our training on the machine learning based approach of (Hermjakob & Mooney, 1997), allowing however unrestricted text and deriving the parse action sequences required for training from a treebank. The basic mechanism for parsing text into a shallow semantic representation is a shift-reduce type parser (Marcus, 1980) that breaks parsing into an ordered sequence of small and manageable parse actions. Figure 2 shows a typical reduce action. The key task of machine learning then is to learn to predict which parse action to perform next.

Two key advantages of this type of deterministic parsing are that its linear run-time complexity with respect to sentence length makes the parser very fast, and that the parser is very robust in that it produces a parse tree for every input sentence.

Figure 3 shows the overall architecture of parser training. From the treebank, we first automatically generate a parse action sequence. Then, for every step in the parse action sequence, typically several dozens per sentence, we automatically compute the value for every feature in the feature set, add on the parse action as the proper classification of the parse action example, and then feed these examples into a machine learning program, for which we use an extension of decision trees (Quinlan, 1986; Hermjakob & Mooney, 1997).

We built our parser incrementally. Starting with a small set of syntactic features that are useful across all languages, early training and testing runs reveal machine learning conflict sets and parsing errors that point to additionally required features and possibly also additional background knowledge. A conflict set is a set of training examples that have identical values for all features, yet differ in their classification (= parse action). Machine learning can therefore not possibly learn how to handle all examples correctly. This is typically resolved by adding an additional feature that differentiates between the examples in a linguistically relevant way.

Even treebanking benefits from an incremental approach. Trained on more and more sentences, and at the same time with also more and more features, parser quality improves, so that the parser as a treebanking tool has to be corrected less and less frequently, thereby accelerating the treebanking process.

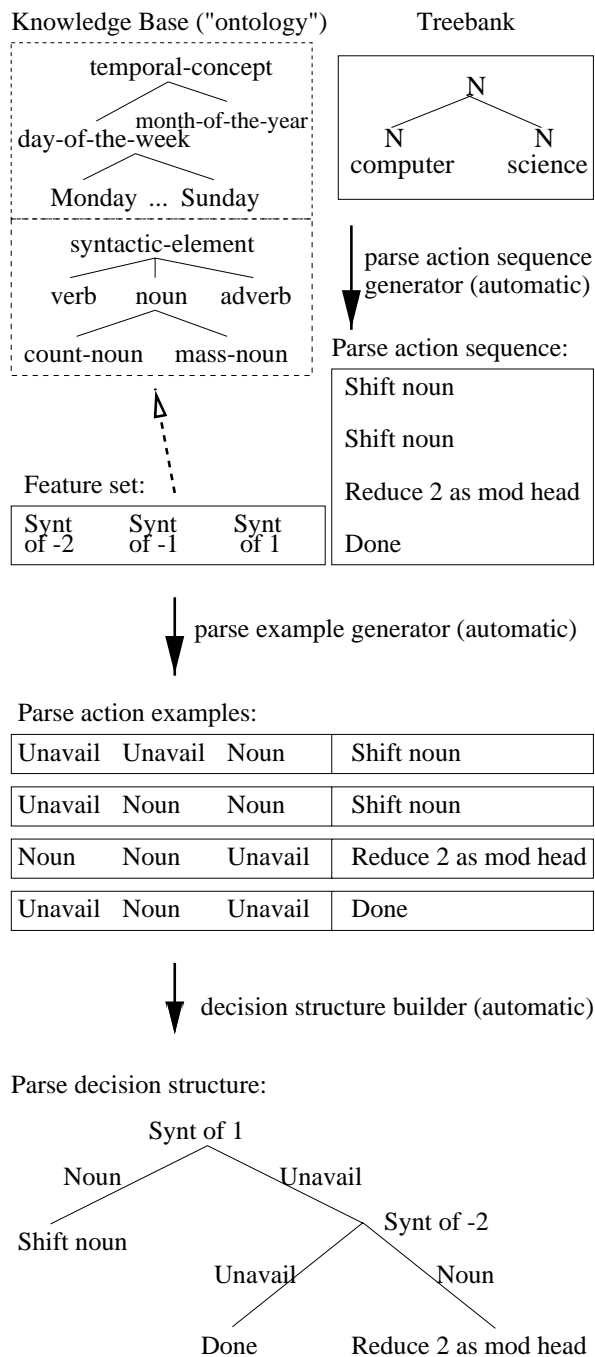


Figure 3: Derivation of the parser from a treebank and a feature set. The resulting parser has the form of a decision structure, an extension of decision trees. Given a seen or unseen sentence in form of a list of words, the decision structure keeps selecting the next parse action until a single parse tree covering the entire sentence has been built.

4.1 Special Adaptation for Korean

The segmenter and morphological analyzer KMA returns a list of alternatives for each *eojeol*. However, the alternatives are not atomic but rather two-level constituents, or mini-trees. Consider for example the following four¹ alternatives for the *eojeol* ‘31il’ (the 31st day of a month):

```
31/NUMERAL + i/SUFFIX-NOUN + l/OBJ-CASE-PRT
31/NUMERAL + i/NUMERAL + l/OBJ-CASE-PRT
31/NUMERAL + il/UNIT-NOUN
31/NUMERAL + il/REGULAR-NOUN
```

The analyzer divides ‘31il’ into groups with varying number of sub-components with different parts of speech. When shifting in an element, the parser has to decide which one to pick, the third one in this case, using context of course.

The module generating parse action sequences from a tree needs special split and merge operations for cases where the correct segmentation is not offered as a choice at all. To make things a little ugly, these splits can not only occur in the middle of a leaf constituent, but even in the middle of a character that might have been contracted from two characters, each with its own meaning.

5 Chosun Newspaper Experiments

Table 1 presents evaluation results with the number of training sentences varying from 32 to 1024 and with the remaining 163 sentences of the treebank used for testing.

Precision:

$$\frac{\text{number of correct constituents in system parse}}{\text{number of constituents in system parse}}$$

Recall:

$$\frac{\text{number of correct constituents in system parse}}{\text{number of constituents in logged parse}}$$

Crossing brackets: number of constituents which violate constituent boundaries with a constituent in the logged parse. **Labeled precision/recall** measures not only structural correctness, but also the correctness of the syntactic label. **Correct operations** measures the number of correct operations during a parse that is continuously corrected based on the logged sequence; it measures the core machine learning algorithm performance in isolation. A sentence has a correct **operating sequence**, if the system fully predicts the logged parse action sequence, and a correct **structure and labeling**, if the structure and syntactic labeling of the final system parse of a sentence is 100% correct, regardless of the operations leading to it.

Figures 4 and 5 plot the learning curves for two key metrics. While both curves are clearly heading

¹KMA actually produces 10 different alternatives in this case, of which only four are shown here.

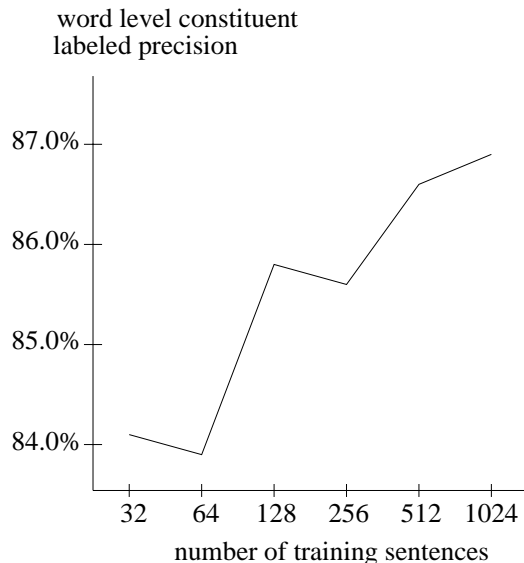


Figure 4: Learning curve for labeled precision corresponding to table 1

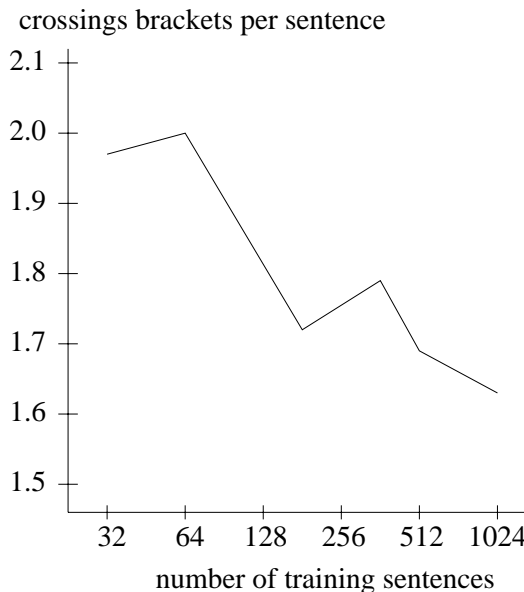


Figure 5: Learning curve for crossing brackets per sentence corresponding to table 1

in the right direction, up for precision, and down for crossing brackets, their appearance is somewhat jagged. For smaller data sets like in our case, this can often be avoided by running an n-fold cross validation test. However, we decided not to do so, because many training sentences were also used for feature set and background knowledge development

Training sentences	32	64	128	256	512	1024
Precision	88.6%	88.1%	90.0%	89.6%	90.7%	91.0%
Recall	87.3%	87.4%	89.2%	89.1%	89.6%	89.8%
Labeled precision	84.1%	83.9%	85.8%	85.6%	86.7%	86.9%
Labeled recall	81.2%	81.9%	83.6%	83.6%	84.7%	85.0%
Tagging accuracy	94.3%	92.9%	93.9%	93.4%	94.0%	94.2%
Crossings/sentence	1.97	2.00	1.72	1.79	1.69	1.63
0 crossings	27.6%	35.0%	38.7%	40.5%	43.6%	42.9%
≤ 1 crossing	56.4%	58.9%	63.2%	59.5%	64.4%	62.6%
≤ 2 crossings	70.6%	72.4%	73.0%	71.8%	73.0%	74.2%
≤ 3 crossings	81.0%	81.6%	82.2%	81.6%	82.2%	83.4%
≤ 4 crossings	88.3%	84.0%	91.4%	89.0%	90.8%	89.6%
Correct operations	63.0%	68.3%	71.5%	73.4%	75.0%	76.3%
Operation Sequence	2.5%	6.1%	8.0%	8.6%	11.0%	7.4%
Structure&Label	5.5%	12.9%	11.7%	16.0%	19.0%	16.0%

Table 1: Evaluation results with varying number of training sentences

as well as for intermediate inspection, and therefore might have unduly influenced the evaluation.

5.1 Tagging accuracy

A particularly striking number is the tagging accuracy, 94.2%, which is dramatically below the equivalent 98% to 99% range for a good English or Japanese parser. In a Korean sentence, only larger constituents that typically span several words are separated by spaces, and even then not consistently, so that segmentation errors are a major source for tagging problems (as it is to some degree however also for Japanese²). We found that the segmentation part of KMA sometimes still struggles with relatively simple issues like punctuation, proposing for example words that contain a parenthesis in the middle of standard alphabetic characters. We have corrected some of these problems by pre- and post-processing the results of KMA, but believe that there is still a significant potential for further improvement.

In order to assess the impact of the relatively low tagging accuracy, we conducted experiments that simulated a perfect tagger by initializing the parser with the correctly segmented, morphologically analyzed and tagged sentence according to the treebank.

By construction, the tagging accuracy in table 2 rises to 100%. Since the segmenter/tagger returns not just atomic but rather two-level constituents, the precision and recall values benefit particularly strongly, possibly inflating the improvements for these metrics, but other metrics like crossing brackets per sentence show substantial gains as well. Thus we believe that refined pre-parsing tools, as they are

²While Japanese does not use spaces at all, script changes between *kanji*, *hiragana*, and *katakana* provide a lot of segmentation guidance. Modern Korean, however, almost exclusively uses only a single phonetic script.

Segmentation/ Tagging ("seg/tag")	Regular seg/tag as implemented	Simulating perfect seg/tag
Labeled precision	86.9%	93.4%
Labeled recall	85.0%	92.9%
Tagging accuracy	94.2%	100.0%
Crossings/sentence	1.63	1.13
0 crossings	42.9%	48.5%
≤ 2 crossings	74.2%	85.3%
Structure&Label	16.0%	28.8%

Table 2: Impact of segmentation/tagging errors

in the process of becoming available for Korean, will greatly improve parsing accuracy.

However, for true low density languages, such high quality preprocessors are probably not available so that our experimental scenario might be more realistic for those conditions. On the other hand, some low density languages like for example Tetun, the principal indigenous language of East Timor, are based on the Latin alphabet, separate words by spaces and have relatively little inflection, and therefore make morphological analysis and segmentation relatively simple.

6 Treebank Consistency Checking

It is difficult to maintain a high treebank quality. When training on a small treebank, this is particularly important, because there is not enough data to allow generous pruning.

Treebanking is done by humans and humans err. Even with annotation guidelines there are often additional inconsistencies when there are several annotators. In the Penn Treebank (Marcus, 1993) for example, the word *ago* as in ‘two years ago’, is tagged

414 times as an adverb and 150 times as a preposition.

In many treebanking efforts, basic taggers and parsers suggest parts of speech and tree structures that can be accepted or corrected, typically speeding up the treebanking effort considerably. However, incorrect defaults can easily slip through, leaving blatant inconsistencies like the one where the constituent ‘that’ as in ‘the dog that bit her’ is treebanked as a noun phrase containing a conjunction (as opposed to a pronoun).

From the very beginning of treebanking, we have therefore passed all trees to be added to the treebank through a consistency checker that looks for any suspicious patterns in the new tree. For every type of phrase, the consistency checker draws on a list of acceptable patterns in a BNF style notation. While this consistency checking certainly does not guarantee to find all errors, and can produce false alarms when encountering rare but legitimate constructions, we have found it a very useful tool to maintain treebank quality from the very beginning, easily offsetting the about three man days that it took to adapt the consistency checker to Korean.

For a number of typical errors, we extended the checker to automatically correct errors for which this could be done safely, or, alternatively, suggest a likely correction for errors and prompt for confirmation/correction by the treebanker.

7 Conclusions

Comparisons with related work are unfortunately very problematic, because the corpora are different and are sometimes not even described in other work. In most cases Korean research groups also use other evaluation metrics, particularly dependency accuracy, which is often used in dependency structure approaches. Training on about 40,000 sentences (Collins, 1997) achieves a crossing brackets rate of 1.07, a better value than our 1.63 value for regular parsing or the 1.13 value assuming perfect segmentation/tagging, but even for similar text types, comparisons across languages are of course problematic.

It is clear to us that with more training sentences, and with more features and background knowledge to better leverage the increased number of training sentences, accuracy rates can still be improved significantly. But we believe that the reduction of parser development time from two years or more down to three months is in many cases already very valuable, even if the accuracy has not ‘maxed out’ yet. And given the experience we have gained from this project, we hope this research to be only a first step to an even steeper development time reduction. A particularly promising research direction for this is to harness knowledge and training resources across languages.

Acknowledgments

I would like to thank Kyoosung Lee for installing, improving and connecting Korean pre-processing tools like segmenter and tagger as well as starting the treebanking, and Mina Lee, who did most of the treebanking.

References

- M. J. Collins. 1997. Three Generative, Lexicalised Models for Statistical Parsing. In *35th Proceedings of the ACL*, pages 16–23.
- U. Hermjakob and R. J. Mooney. 1997. Learning Parse and Translation Decisions From Examples With Rich Context. In *35th Proceedings of the ACL*, pages 482–489.
URL: file://ftp.cs.utexas.edu/pub/mooney/papers/context-acl-97.ps.Z
- U. Hermjakob. 1997. *Learning Parse and Translation Decisions From Examples With Rich Context*. Ph.D. thesis, University of Texas at Austin, Dept. of Computer Sciences TR 97-12.
URL: file://ftp.cs.utexas.edu/pub/mooney/papers/hermjakob-dissertation-97.ps.Z
- Geunbae Lee, Jong-Hyeok Lee, and Hyuncheol Rho. 1997. Natural Language Processing for Session-Based Information Retrieval Interface on the Web. In *Proceedings of IJCAI-97 workshop on AI in digital libraries*, pages 43–48.
- M. P. Marcus. 1980. *A Theory of Syntactic Recognition for Natural Language*. MIT Press.
- M. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), pages 313–330.
- J. R. Quinlan. 1993. *C4.5 Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, California.
- K. J. Seo, K. C. Nam, and K. S. Choi. 1998. A Probabilistic Model for Dependency Parsing Considering Ascending Dependencies. *Journal of Literary and Linguistic Computing*, Vol 13(2).
- Juntae Yoon, Seonho Kim, and Mansuk Song. 1997. New Parsing Method Using Global Association Table. In *Proc. of the International Workshop on Parsing Technology*.