Acta Cybernetica **17** (2006) 647–659.

# Learning Tree Patterns for Syntactic Parsing

András Hócza*

**Abstract**

This paper presents a method for parsing Hungarian texts using a machine learning approach. The method collects the initial grammar for a learner from an annotated corpus with the help of tree shapes. The PGS algorithm, an improved version of the RGLearn algorithm, was developed and applied to learning tree patterns with various phrase types described by regular expressions. The method also calculates the probability values of the learned tree patterns. The syntactic parser of learned grammar using the Viterbi algorithm performs a quick search for finding the most probable derivation of a sentence. The results were built into an information extraction pipeline.

## 1 Introduction

Syntactic parsing is the process of finding the immediate constituents of a sentence that is a sequence of words. Syntactic parsing is an important part of the field of natural language processing and it is useful for supporting a number of large-scale applications including information extraction, information retrieval, named entity identification, and a variety of text mining applications.

The Hungarian language is customarily defined as an agglutinative, free word order language with a rich morphology. These properties make its full analysis difficult compared to Indo-European languages. Unambiguous markers for the automatic rec-ognition of phrase boundaries do not exist. For example, the right bound of noun phrases could be the nouns as a head, but there is a possibility of omitting noun phrase head using its modifiers only. Determining the left bound of noun phrases is harder than the head, as it could be a determinant element. However, due to the possibility of a recursive insertion, it is not easy to decide which determiner and head belong to-gether. These difficulties mean that it is a quite hard to perform syntactic parsing with expert rules. In many cases the decision of annotators is based on semantic features rather than syntactic or structural ones.

An efficient solution for the problem of syntactic parsing might be the application of machine learning methods, but this requires a large number of training

*University of Szeged, Department of Informatics, H-6720 Szeged, Árpád tér 2., Hungary, E-mail: hocza@inf.u-szeged.hu

and test examples of annotated syntax trees. Since the Szeged Corpus[1] [3] became available, new methods have begun to be developed for syntactically parsing Hungarian sentences. The corpus contains texts from five different topic areas and currently has about 1.2 million word entries, 145 thousand different word forms, and an additional 225 thousand punctuation marks. The corpus contains manually POS tagged and disambiguated sentences and it was parsed by annotators who marked noun phrase structures and various clause structures.

After the completion of the annotation work the Szeged Corpus was then used for training and testing machine learning algorithms to retrieve syntactic grammars. For this the PGS (Pattern Generalization and Specialization) algorithm is introduced here, an improved version of the RGLearn algorithm [6] for learning syntactic tree patterns. The initial grammar for learner is collected from the Szeged Corpus using tree shapes to disassemble syntactic trees. Probability values of learned tree patterns are also computed. Based on this probability grammar the parsing algorithm incorporated in the Viterbi search method [18] can find quickly the most probable derivation of a sentence.

This paper is organized as follows. In Section 2 there is a review of related works and a description of efforts made by Hungarian researchers. Section 3 introduces a method used for collecting initial grammar for a learner from a large annotated corpus. Section 4 then presents the method used for learning grammar from an annotated corpus and extending this grammar with probability values using statistics. Section 5 introduces our method of syntactic parsing. After, Section 6 presents the test results we obtained. Finally, conclusions and suggestions for future study are given in Section 7.

## 2 Related works

Several authors have published results of syntactic parsing mainly for English. Generally the performance is measured with three scores. First, with a percentage of detected noun phrases that are correct (precision). Second, with a percentage of noun phrases in the data that is found via the classifier (recall). And third, with the $F_{\beta=1}$ rate which is equal to 2*precision*recall/(precision*recall). The latter rate has been used as the target for optimization.

Abney [2] proposed an approach which starts by finding correlated chunks of words. Ramshaw and Marcus [12] built a chunker by applying transformation-based learning ($F_{\beta=1}$=92.0). They applied their method to two segments of the Penn Treebank [10] and these are still being used as benchmark data sets. Argamon [4] uses memory-based sequence learning ($F_{\beta=1}$=91.6) for recognizing both NP chunks and VP chunks. Tjong Kim Sang and Veenstra [15] introduced cascaded chunking ($F_{\beta=1}$=92.37). The novel approaches attain good accuracies using a system combination. Tjong Kim Sang [14] utilized a combination of five classifiers ($F_{\beta=1}$=93.26).

[1] Magyar Távirati Iroda, `http://www.mti.hu`

Up to now there has been no good-quality syntactic parser available for Hungarian. Benchmark data sets for correctly comparing results on Hungarian do not exist yet either. The HumorESK syntactic parser [8] developed by MorphoLogic Ltd uses attribute grammar, assigning feature structures to symbols. The grammar part employed in the parser was made by linguistic experts. Another report on the ongoing work of a Hungarian noun phrase recognition parser [17] is based on the idea of Abney [1] using a cascaded regular grammar. The input to the grammar was a morpho-syntactically annotated text using a scaled-down version of the annotation scheme developed for the Hungarian National Corpus [16]. The grammar was developed by linguistic experts with the help of the CLaRK corpus development system [7] and it was tested in a short text of annotated sentences ($F_{\beta=1}$=58.78). The idea of using cascaded grammars seems beneficial, this technique being used in all Hungarian parser developments. A noun phrase recognition parser [6] applied machine learning methods to produce a grammar of noun phrase tree patterns from an annotated corpus ($F_{\beta=1}$=83.11).

# 3 Producing initial grammar for machine learning

Preprocessing provide training and test examples for machine learning. This process collects examples from the annotated corpus and transforms information into that needed for the learning problem. According to our approach the collected examples are tree patterns that can be used as grammar in a syntactic parser to help rebuild syntax tree of sentences. The role of machine learning is the generalization of this grammar to achieve good accuracy scores on unknown sentences as well.

## 3.1 Data source for training and evaluation

In order to perform well and learn from the various *Natural Language Processing* (NLP) tasks and achieve a sufficient standard of *Information Extraction* (IE), an adequately large corpus had to be collected that serves as the training database. While setting up various NLP projects, a relatively large corpus of varous types of texts was collected: the Szeged Corpus [3]. For demonstration purposes in the above-mentioned projects the authors chose a collection of short business news items issued by the Hungarian News Agency[2]. The chosen 6453 articles form part of the Szeged Corpus and relate to companies, financial and business life. The Szeged Corpus contains 1.2 million words, 225 thousand punctuation marks, and comes in an XML format using the TEIXLite DTD. The first version of the corpus contains texts from five topic areas, roughly 200 thousand words each, meaning a text database of some 1 million words. The second version was extended with a sample of texts of business news totalling another 200 thousand words. The texts are divided into sections, paragraphs, sentences and word entries.

Initially, corpus words were morpho-syntactically analysed with the help of the Humor [11] automatic pre-processor and then manually POS tagged by linguistic

[2]MTI, Magyar Távirati Iroda (http://www.mti.hu), "Eco" service.

experts. The Hungarian version of the internationally acknowledged MSD (Morpho-Syntactic Description) scheme [5] was used for encoding words. Due to the fact that the MSD encoding scheme is extremely detailed (one label can store morphological information on up to 17 positions), there are a lots of ambiguous cases, i.e. roughly every second word of the corpus is ambiguous. Disambiguation therefore required accurate and detailed work. Actually 64 person-months of manual annotation was needed for this. Currently all possible labels as well as the selected ones are stored in the corpus. About 1800 different MSD labels are used in the annotated corpus. The MSD label corresponds to the part-of-speech determined attribute, and specific characters in each position indicate the value for that attribute.

For example, the MSD label Nc-pa can be understood as

| | |
|---|---|
| POS: | **N**oun, |
| Type: | **c**ommon, |
| Gender: | **-** (not applicable to Hungarian), |
| Number: | **p**lural, |
| Case: | **a**ccusative. |

All the texts of Szeged Corpus were parsed manually, that is annotators marked various phrase structures. The extensive and accurate manual annotation of the texts, which required 124 person-months of manual work, is a good feature of the corpus. The syntax trees of annotated sentences contain various type of phrases, shown by following list:

| | |
|---|---|
| Noun phrase (NP) | Verb prefix (PREVERB) |
| Adjective phrase (ADJP) | Conjunction (C) |
| Adverb phrase (ADVP) | Pronoun phrase (PP) |
| Verb phrase (VP) | Clause (CP) |
| Infinitive(INF) | Sentence (S) |
| Negative (NEG) | |

## 3.2 Converting syntax trees to set of rules

Figure 1 shows a one-level pattern retrieving process. The tree disassembly is executed bottom-up, step by step, where a step consists of following substeps:

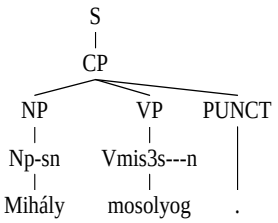
1. The exploration of base phrases (innermost phrases).
2. The storing base phrases.
3. Substituting base phrases with appropriate terminal symbols.

The tree disassembly process is completed when only the topmost symbol (S) remains. The stored phrases are actually rules, because there is a possibility of reconstructing the original syntax tree with their help.

## 3.3 Tree disassembly with tree shapes

The disadvantage of one-level context-free rules is that important information disappears from the contextual conditions of rule applications. This can cause errors

Syntax tree of a short sentence:




Bracketed sentence:
($_S$($_{CP}$($_{NP}$Np-sn|Mihály) ($_{VP}$Vmis3s---n|mosolyog)PUNCT|.))

One level patterns:
($_{NP}$Np-sn|Mihály)
($_{VP}$Vmis3s---n|mosolyog)
($_{CP}$ NP VP PUNCT|.)
($_S$ CP)

Retrieved grammar:
NP → Np-sn|Mihály
VP → Vmis3s---n|mosolyog
CP → NP VP PUNCT|.
S → CP

Figure 1: Short example for tree disassembling (Mihály is smiling.)

in syntax parsing especially in the case of short rules. For example in Figure 1 there is a short rule: *NP → Np-sn|Mihály.* In a more complex sentence (e.g. *the friendly Mihály is smiling*) the application of this rule causes an error because the noun phrase consists of not only Mihály (but *the friendly Mihály*). For this rule we need a context to know whether its application is error-free. This derivation may not be completed with an S symbol and in this case the derivation is dropped, but due to short context-free rules there is a chance that if we take longer sentences many false derivations will appears in the derivation forest, making the runtime longer and the selection of the best tree harder.

Our solution for reducing problems with context-free rules is the extraction of *subtree patterns* or simply *tree patterns*. These patterns contain more than one node in hierarchical structures i.e. they are trees. The description of tree patterns includes word positions with their stems and lexical codes and bracketed phrases. Applying tree patterns instead of one-level context-free rules, short phrases are placed together with their contexts, so the derivation with tree patterns reduces ambiguities and raises the accuracy score. But using patterns brings another question: what size of trees are useful? Big trees are too wide or too deep, or both, and

too specific. Grammars with over-specific rules can also result in poor accuracy scores on unknown texts because of missing covering rules. Hence not only short rules must be eliminated, but big trees as well.

To determine the proper size of trees extracted from a training corpus *tree shape types* are used. Tree shape types are a general form of subtrees that are defined with some property in their description. These types were chosen by linguistic experts studying which syntactic structures appear most often in annotated sentences. In the end the following two tree shape types were found:

1. "Hole":
   - Contains at least two terminals
   - It can be divided into one deepen and one risen parts
   - It has a recursive structure in Hungarian, the inside tree appended or prefixed with terminals
   - Example (Eng. 'on the edge of a terrible whirlpool'):

```
(NP
    (NP
        Ti|egy
            (ADJP
                Afp-sn|rettenetes
            )
        Nc-sn|örvény
    )
    Nc-sp—s3|szélén
)
```

2. "String":
   - Its depth is 2
   - It may contains more subtrees, terminals or substituted nodes
   - The depth of its subtrees is 1
   - It can be enumerated in Hungarian, the small trees, terminals or substituted nodes follow each other and these are included by a single phrase
   - Example (Mihály és Erzsi, 'Michael and Liz'):

```
(NP
    (NP
        Np-sn|Mihály
    )
    (C
        Ccsw|és
    )
    (NP
        Np-sn|Erzsi
    )
)
```

# 4 Learning tree patterns

In this section the learning task of syntactic tree patterns are described, which contains the preprocessing of training data, and the generalization and specialization of tree patterns. An improved version of the RGLearn algorithm [6] called PGS (*Pattern Generalization and Specialization*) was used as a tree pattern learner. The novelty of PGS is the use of $\lambda$ parameters which have an influence on the quality of learned tree patterns. The pattern unification method and the search method for the best patterns were also modified.

## 4.1 Preprocessing of training data

The initial step for generating training data is to collect syntactic tree patterns from an annotated training corpus. The complete syntax tree of a sentence must be divided into separate trees and a cascade of tree building rules to help prepare the parser in the reconstruction of it. In parsing, using context-free grammars has a lot of advantages, but the conditions of pattern usage may completely disappear. Some structural information can be retained if tree patterns are used. To generate cascaded grammar, linguistic experts have defined the following processing levels for the Hungarian language:

- Short tree patterns of nominal, adjectival, adverbial and pronominal phrases.
- Recursive patterns of nominal, adjectival, adverbial and pronominal phrases.
- Recursive patterns of verb phrases.
- Recursive patterns of sub-sentences.

## 4.2 The generalization of tree patterns

Using the collected tree patterns the syntactic parser is able to reconstruct the tree structure of training sentences. In order to perform the syntactic parsing of an unknown text to a fair accuracy, the collected tree patterns must, however, be generalized. Generalization means that the lexical attributes of each tag are neglected except for the POS codes. In this phase the learning problem is transformed into a classification problem. Namely the following question should be answered: which set of lexical attributes provides the best result for the decision problem of tree pattern matching i.e. a given tree structure covers a given example or not. To support the learner, positive and negative examples are collected from training sets for each tree type. The example in Figure 2 shows the complete tree pattern learning process.

## 4.3 Specialization of tree patterns

Negative examples are bad classifications of generalized tree patterns and they must be eliminated. Therefore specialization selects each possible lexical attribute from

Sentence parts (examples):
1: . . . $({}_{NP}Tf({}_{ADJP}Afp-sn)Np-sn)$ . . .
2: . . . $({}_{NP}Tf({}_{NP}Afp-sn)Nc-pa)$ . . .
3: . . . $({}_{NP}Ti({}_{NP}Afs-sn))({}_{NP}Nc-s2)$ . . .
4: . . . $({}_{NP}Tf({}_{ADJP}Afp-sn)Nc-sn)$ . . .
5: . . . $({}_{NP}Tf({}_{ADJP}Afp-sn)({}_{ADJP}Afp-sn))$ . . .

One of four possible generalized pattern: $({}_{NP}T*({}_{ADJP}A*)N*)$
Coverage: positive $\{1,4\}$, negative $\{2,3\}$, uncovered $\{5\}$
Specialized pattern: $({}_{NP}T*({}_{ADJP}A*)N???n)$
Coverage: positive $\{1,4\}$, negative $\{\}$, uncovered $\{2,3,5\}$

Notations:
The first letter of morpho-syntactic codes is the part of speech
Determiner: T*, Adjective: A*, Noun: N*
A letter of any kind: ?, One or more letters of any kind: *

Figure 2: A tree pattern learning example.

positive examples making new tree patterns, and tries to find the best tree patterns via unification.

The initial step of specialization generates all possible new tree patterns by extending generalized tree patterns with exactly one attribute from the covered positive examples. The next steps of specialization extend the set of tree patterns with all possible new tree patterns by a combination of each pair of tree patterns. The combination of two tree patterns means the union of their lexical attributes. To avoid the exponential growth of a tree pattern set weak tree patterns are excluded by applying error statistics on positive and negative examples. The following score of a given tree pattern is used as the target for maximization:

$$score = \lambda_1 * (pos - neg)/pos + \lambda_2 * (pos - neg)/(pos + neg)$$

where *pos* is the number of covered positive examples, *neg* is the number of covered negative examples and $\lambda_1 + \lambda_2 = 1$.

Fruitful unifications dramatically decrease the negative coverage. The score maximization operates in parallel on every positive example. A new tree pattern is stored only if a covered positive example exists where the score of the new tree pattern is greater than the previous maximum value. Specialization ends when the current step did not improve any maximum value.

Appropriate setting of $\lambda$ factors in linear combination can provide the optimal tree pattern set. A greater $\lambda_1$ may result in tree patterns with high coverage, while a greater $\lambda_2$ may resulting a higher accuracy but there is a possibility of low tree patterns appearing with a low coverage.

### 4.4 Producing probabilistic grammar

The syntax tree of a sentence can generally be derived in different ways, especially when using a large grammar. The choice of a best derivation from the derivation forest requires additional information. One of the possible ways of doing this is a making of PCFG (Probability Context Free Grammar). Using a PCFG, the probability of a derivation is the product of probabilities of the applied rules, but the product may be replaced by other operators, e.g. a max operator. After evaluating each derivation, the derivation with a higher probability will selected. It is described with the following formulas:

$$P(Derivation) = \prod_{T \to \beta \in Derivation} P(T \to \beta | T) \quad (1)$$

$$Best = \underset{Derivation \in Forest}{\operatorname{argmax}} P(Derivation) \quad (2)$$

The conditional probability of a rule is computed with the following formula of Maximum Likelihood Estimation:

$$P(T \to \beta | T) = \frac{Count(T \to \beta)}{Count(T)} \quad (3)$$

Finally, the last formula shows that the conversion of a CFG to a PCFG is based on rule application statistics on the training data, namely, counting the proper coverage of each rule and counting the node labels in the annotated syntax trees. This method is applicable for tree patterns as well. Counting the coverage of tree patterns is the same as counting coverage of one level rules. The left symbol for a CFG rule of tree patterns is a root of the subtree:

Tree patterns:
($_{NP}$($_{NP}$ T* ($_{ADJP}$ A* ) N??s? ) N???????s3 )
($_{NP}$($_{NP}$ N* ($_{C}$ C* ) N* ) )
Rule form (brackets indicate the structural information):
NP → ($_{NP}$ T* ($_{ADJP}$ A* ) N??s? ) N???????s3
NP → ($_{NP}$ N* ($_{C}$ C* ) N* )

## 5 Syntactic parsing

The main task of the syntactic parser is to find the most likely parse tree for input sentences. Input data contains disambiguated POS tag labels and it may come from the *Szeged Corpus*during the testing phase of the method, or it may be provided by a POS tagger tool [9] as a practical application of the method. There are natural language processing modules used to determine various linguistic features for syntactic parsing when the process starts with plain text: tokenization, sentence segmentation, morpho-syntactic analysis and part-of-speech (POS) tagging.

Parsing with a PCFG can be done in polynomial time - but disambiguation - namely the selection of best possible parse tree from the parse forest, is an NP-hard

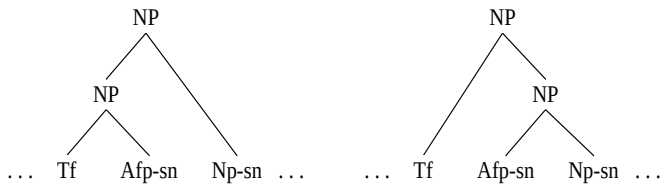



Figure 3: Two different subderivations for the same part of a sentence.

problem. Sima'an [13] demonstrated that there is no deterministic polynomial time algorithm for finding the most probable parse of a sentence. Owing to this fact, it is not efficient if the parser determines all possible derivations with probabilities, and then selects the best one. Hence, the Viterbi algorithm [18] is applied in our parser to find the most probable derivation, because it can perform it in cubic time. The basic idea behind the Viterbi approach is the elimination of low probability subderivations in a bottom-up fashion. Two different subderivations for the same part of sentence and with the same root can both be included by derivations in the same way. Thus, if one of these two subderivations has a lower probability, it will be eliminated. The selection situation is illustrated by a short example in Figure 3.

The Viterbi elimination method is applicable for higher levels in bottom-up parsing and finally for the selection among S roots of the derivation forest. There are a lot of low probability subderivations that are pruned through parsing to speed up the process. The syntactic parsing of a sentence involves the following:

```
Take a POS tagged sentence
    BEST_TREE = nil
    RULE_LEVEL = 0
    repeat
        derive with each tree patterns from Cascade_Set ( RULE_LEVEL )
        foreach NEW_NODE do // Viterbi elimination for new nodes
            if exist SAME_NODE as NEW_NODE then
                Eliminate_Weaker ( NEW_NODE , SAME_NODE )
        if exist ROOT_NODE == S then
            BEST_TREE = Tree ( ROOT_NODE )
        RULE_LEVEL = RULE_LEVEL + 1
    until (applicable tree pattern exist)
```

Note: depending to the rule set used, the full parse tree with an S root does not always exist for an arbitrary input sentence.

Table 1: Test results on the two corpus domains.

| **Text category** | **Precision** | **Recall** | $F_{\beta=1}$ |
|---|---|---|---|
| Corpus version 1.0 | 82.27% | 79.35% | 80.78% |
| Business news extension | 85.83% | 81.46% | 83.59% |

# 6 Experiments

The training and evaluation datasets were taken from various parts of the Szeged Corpus. One of the domains we examined was the first version of the corpus containing texts from five different topic areas and the second domain was the business news extension of the Szeged Corpus. The training ad test was performed using 10-fold cross-validation. The sentences of domains were randomly divided into ten parts. Then ten different train (90%) and test (10%) sets were pieced together from parts and learning and testing was performed ten times. The average of the results on ten test sets is shown in Table 1.

In general, the precision score is higher than the recall, because of the unknown structures. The better results in business news extension may be caused by the different characteristics of domains. The business news part contains relatively homogeneous texts with often repeated phrases and idioms. The first domain is more heterogeneous in its five topics. Based on the experiments, the syntactic parser - as an element of a natural language processing pipeline - provided enough information for information extraction.

# 7 Summary and future work

In this paper a general tree pattern learning method for syntactic parsing was presented. The initial grammar for learner was collected from a large corpus domain using tree shapes to disassemble a syntactic tree of annotated sentences. The PGS algorithm presented, an improved version of the RGLearn machine learning algorithm, performs generalization and specialization to produce grammar. Probability values of learned tree patterns were also computed. For the parsing algorithm, Viterbi searching was used to speed up finding the most probable derivation. Based on results the syntactic parser, as an element of a natural language processing pipeline, provides sufficiently rich information to support information extraction.

In the future, in parallel to the development of the Szeged Corpus, the developing machine learning methods are intended for the same problems as syntactic parsing i.e. named entity recognition and semantic frame identification. Improving the results of syntactic parser building in other methods, e.g. system combinations and the usage of ontological information that could be the extension of a morpho-

syntactic description is also planned. The system described here will be primarily applied to the business news domain. In the 6th Framework Programme of the European Commission, an international research consortium is planning to develop a multilingual IE system for the above-mentioned two domains.

## References

[1] Abney, S. Partial parsing via finite-state cascades. In *Proceedings of ESSLLI'96 Robust Parsing Workshop*, pages 1–8, 1996.

[2] Abney, S. *Parsing by chunks*. Kluwer Academic Publishers, 2003.

[3] Alexin, Z., Csirik, J., Gyimóthy, T., Bibok, K., Hatvani, Cs., Prószéky, G., and Tihanyi, L. Manually annotated hungarian corpus. In *Proceedings of the Research Note Sessions of the 10th Conference of the European Chapter of the Association for Computational Linguistics EACL03*, pages 53–56. Budapest, Hungary, 2003.

[4] Argamon, S., Dagan, I., and Krymolowski, Y. A memory-based approach to learning shallow natural language patterns. In *Proceedings of 36th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 67–73. Montreal, 1998.

[5] Erjavec, T. and Monachini, M., ed. *Specification and Notation for Lexicon Encoding*. Copernicus project 106 MULTEXT-EAST, Work Package WP1 - Task 1.1 Deliverable D1.1F., (1997.

[6] Hócza, A. Noun phrase recognition with tree patterns. *Acta Cybernetica, vol 16*, pages 611–623, 2004.

[7] K., Simov. Clark - an xml-based system for corpora development. In *Proceedings of the Corpus Linguistics 2001 Conference*, pages 553–560. Lancaster, 2001.

[8] Kis, B., Naszódy, M., and Prószéky, G. Complex hungarian syntactic parser system. In *Proceedings of the MSZNY 2003*, pages 145–151. Szeged, Hungary, 2003.

[9] Kuba, A., Bakota, T., Hócza, A., and Oravecz, Cs. Comparing different pos-tagging techniques for hungarian. In *Proceedings of the MSZNY 2003*, pages 16–23. Szeged, Hungary, 2003.

[10] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. Building a large annotated corpus of english: the penn treebank. *Association for Computational Linguistics*, 1993.

[11] Prószéky, G. and Kis, B. A unification-based approach to morpho-syntactic parsing of agglutinative and other (highly) inflectional languages. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 261–268. College Park, Maryland, USA, 1999.

[12] Ramshaw, L. A. and Marcus, M. P. Text chunking using transformational-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*. Association for Computational Linguistics, 1995.

[13] Sima'an, K. Computational complexity of probabilistic disambiguation by means of tree grammars. In *Proceedings of COLING-96*. Copenhagen, 1999.

[14] Tjong Kim Sang, E. F. Noun phrase recognition by system combination. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 50–55. Seattle, 2000.

[15] Tjong Kim Sang, E. F. and Veenstra, J. Representing text chunks. In *Proceedings of EACL '99 Conference*. Association for Computational Linguistics, 1999.

[16] Váradi, T. The hungarian national corpus. In *Proceedings of the Second International Conference on Language Resources and Evaluation LREC2002*, pages 385–389. Las Palmas de Gran Canaria, 2002.

[17] Váradi, T. Shallow parsing of hungarian business news. In *Proceedings of the Corpus Linguistics 2003 Conference*, pages 845–851. Lancaster, 2003.

[18] Viterbi, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans Information Theory, vol IT-13*, pages 260–269, 1967.