Automatic Evaluation of Robustness and Degradation in Tagging and Parsing

Johnny Bigert, Ola Knutsson, Jonas Sjöbergh
Department of Numerical Analysis and Computer Science
Royal Institute of Technology, Sweden
{johnny,knutsson,jsh}@nada.kth.se

Keywords: automatic evaluation, robustness, spelling errors, tagging, shallow parsing

Abstract

We address the topic of automatic evaluation of robustness and performance degradation in parsing systems. We focus on one aspect of robustness, namely ill-formed sentences and the impact of spelling errors on the different components of a parsing system. We propose an automated framework to evaluate robustness, where ill-formed and noisy data is introduced using an automatic tool and fed to the parsing system. With increasing levels of noise, the performance of a system will inevitably degrade, and the question is to what extent? The experiments show a graceful degradation in performance for both state-of-the-art taggers used and a Swedish shallow parser. The automated nature of the evaluation allows easy and reproducible evaluation of the individual components of a parsing system.

1 Introduction

Comparable and reproducible evaluations and experiments are important foundations for research. Valid and comparable evaluation of NLP-systems is often hard to achieve without a great portion of manual work. Many languages lack resources for evaluation and even for languages with available material, new types of evaluations based on existing material need a lot of manual labor. Furthermore, manual interference with the evaluation material may in fact decrease the validity of the evaluation.

In this paper we present an automatic evaluation method focusing on the robustness of parsers for syntactic analysis. The robustness of a parser is defined here as robustness against ill-formed input, which is only one aspect as pointed out by Menzel 95. The proposed evaluation technique is fully automatic, with no need for manual annotation or inspection. It relies on a general tool, which introduces different kinds of errors into a text. The errors can be spelling or grammatical errors, but we have focused on spelling errors resulting in non-existing words to avoid some ambiguity problems, as explained later.

To demonstrate the evaluation method, we applied it on a shallow parser for Swedish. The experiments are presented as a glass box evaluation, where the performance of the over-all system is presented as well as the performance of the components, such as part-of-speech taggers. All tests are conducted with various levels of errors introduced, under which the system performance degradation is measured. Since this paper focuses on evaluation methodology, we do not address how the introduced errors affect syntactic structure. Nevertheless, automatic evaluation of the effects on syntactic structure is indeed an interesting topic for future work.

The main contribution of this paper is an automatic method for the evaluation of robustness and degradation in tagging and parsing. The evaluation method is useful for determining the performance impact from individual components and thus, is suitable for e.g. on-going development of parsing systems. We argue that introducing noise in text with automatic means is a valid and reliable technique for evaluating a system's robustness.

This paper consists of five parts. To begin with, we discuss evaluation methods for parsing and subsequently, the proposed method for automatic evaluation is presented. In the third part, the shallow parser used in the experiments is briefly outlined. In the fourth section, experimental methodology and the experiments are described. Lastly, we present the results.

1.1 Parser Evaluation

Parsers have been evaluated in the parser community using different methods over the years, from simple test suites to treebanks (for a comprehensive overview see Carroll *et al.* 98). A prerequisite for the evaluation method presented here is an annotated corpus, preferably a treebank. For Swedish, there exist small forests annotated with functional structure (Nivre 02). Unfortu-

nately, none of these were suitable for our experiments, which are based on an annotation scheme for constituency. We adopted the IOB tag set (Ramshaw & Marcus 95) for row-based phrase assignment of Swedish constituents.

Using different text genres for testing is one way of evaluating robustness (see e.g. Li & Roth 01). The problem with such an evaluation is the lack of control of the differences between the genres; is sports harder to parse than economical news? Another way is to compare well-written and proofread texts with more noisy texts. One problem with such a comparison is the difficulty in comparing the two text types without manual analysis of the errors. In the following section we will present a method that compares proofread text and text with different degrees of automatically introduced noise. The method is one way of evaluating a specific kind of robustness of an NLP-system.

2 The Proposed Method

When processing unrestricted natural language data, rare and malformed constructions and spelling errors occur quite frequently. We want to assess the degree to which a parser can handle these difficulties, and to which degree the analysis fails for the context surrounding a deficiency. To this end, we require a source of texts with annotated language errors. One problem is that annotated resources containing errors do not exist for all languages. Furthermore, the texts must also be annotated with parse information to be able to serve as a comparison to the output of a parser. Where data of sufficient size exists, it is often not sufficiently annotated in terms of parse trees. Another problem with annotated errors is the difficulty in determining, for a given error, what the corrected text should be. Normally, we do not know what the intention of the writer was, which introduces further ambiguity in the parse trees.

2.1 Introducing Errors

To avoid the problems mentioned, we decided to start from correct text and from there, introduce errors using an automated tool, MISSPLEL (Bigert *et al.* 03).

MISSPLEL is a general-purpose error introducer, able to introduce most error types produced by humans, such as spelling errors resulting in existing and non-existing words, with or without a change in part-of-speech (PoS) tag, competence errors such as sound-alike errors and context sensitive errors such as feature agreement errors and word order errors.

Most of these error types have the effect of altering the original semantics of the sentence, especially when a word is misspelled, resulting in an existing word. This is indeed a problem since the parse tree of the new sentence may differ from that of the original sentence. Thus, there is a possibility that the output of the parse system is in fact correct even though it differs from the annotated parse tree. We approach this problem by restricting the introduced errors to spelling errors that result in non-existing words only. Hence, the new sentence does not have a straightforward interpretation. Nevertheless, the most plausible interpretation of the new sentence is that of the original text.

During introduction of errors, every word containing alphanumeric characters has an equal chance of being misspelled. Given a word to misspell, we choose a position in the word randomly. To simulate performance errors caused by keyboard mistypes, we choose between insertion, deletion and substitution of a single letter as well as transposition of two letters (Damerau 64). When substituting a letter for another, letters close on the keyboard are more often mistaken than those far apart. The situation is similar for insertion, since this is often caused by pressing two keys at the same time. Thus, keys closer on the keyboard have a higher confusion probability when substituting and inserting. For further details, see Bigert *et al.* 03.

The automatic introduction of spelling errors permits us to choose the percentage of errors in a text. It also allows us to choose any text or text type provided that it is annotated with parse trees. Furthermore, it allows for an n-iteration test to determine how different phrase types degrade with increasing number of errors, and thus, minimizing the influence of chance.

2.2 Evaluation

To gather the necessary data, we used an automated evaluation tool called AUTOEVAL (Bigert *et al.* 03), which is a general purpose evaluation

```
Viktigaste (the most important) APB|NPB  CLB
redskapen (tools)               NPI      CLI
vid (in)                        PPB      CLI
ympning (grafting)              NPB|PPI  CLI
är (is)                         VCB      CLI
annars (normally)               ADVPB    CLI
papper (paper)                  NPB|NPB  CLI
och (and)                       NPI      CLI
penna (pen)                     NPB|NPI  CLI
,                               O        CLB
menade (meant)                  VCB      CLI
han (he)                        NPB      CLI
.                               O        CLI
```

Figure 1: *Example sentence showing the IOB format.*

```
((CL (NP (AP Viktigaste) redskapen)
     (PP vid (NP ympning))
     (VC är)
     (ADVP annars)
     (NP (NP papper) och (NP penna)))
 (CL  ,
     (VC menade)
     (NP han)) . )
```

Figure 2: *The text from Figure 1 in a corresponding bracketing format.*

tool for structured data (e.g. row-based data or XML) configurable using a script language.

The output from the GTA parser was given in the so-called IOB format (Ramshaw & Marcus 95). See Figure 1 and 2 for a sentence with phrase labels and clause boundaries in the IOB and bracketing format, respectively. As an example, `NPB|PPI` means that the beginning (`B`) of an noun phrase (`NP`) is inside (`I`) a prepositional phrase (`PP`). Thus, the rightmost phrase is the topmost node in the corresponding parse tree. The phrase types are explained in Section 3.

Using AutoEval, we gathered information on tag accuracy, full row parse accuracy, clause boundary identification accuracy as well as precision, recall and F-scores for all phrase types.

The statistics for individual phrase types were calculated as follows. Given a phrase type to evaluate, all other phrases were removed. The same was done for the correct, annotated parse, and the results were then compared. The parser was successful if and only if they were identical. For example, we are looking at phrases of type NP. If the correct parse is `APB|NPB|NPI` (an adjective phrase in a noun phrase inside another noun phrase), the parse `NPB|APB|NPI` would be correct since the adjective phrase is ignored, while the parse `APB|NPI|NPI` would be incorrect since the leftmost NP differs.

Since many parsers rely heavily on the performance of a part-of-speech tagger, we include several taggers with different behavior and characteristics. Apart from taggers representing state-of-the-art in part-of-speech tagging, we also include a perfect tagger and a baseline tagger. The perfect tagger does nothing more than adopt the original tags found in the annotated resource. The baseline tagger is constructed to incorporate little or no linguistic knowledge and was included to establish the difficulty of the tagging task.

Parsing different texts may result in different accuracy for the parser at hand. To provide a clue to the inherent difficulty of a text, we require a baseline for the parsing task. The perfect tagger, the baseline tagger and the baseline parser are further discussed in the experiments section.

In the experiments below, we use six error levels (0%, 1%, 2%, 5%, 10%, 20%). For a given error level p, we introduce spelling errors (resulting in non-existing words only) in a fraction p of the words. This procedure is repeated $n = 10$ times to mitigate the influence of chance and to determine the standard deviation of the accuracy and F-scores.

With increasing amounts of erroneous text, the performance of the parser will degrade. In order to be robust against ill-formed and noisy input, we want the accuracy to degrade gracefully with the percentage of errors. That is, for a parser relying heavily on PoS tag information, we aim for the parsing accuracy to degrade equal to or less than the percentage of tagging errors introduced. Of course, this is not feasible for all phrase types. For example, when the infinite marker or verb is misspelled, an infinitival verb phrase will be difficult to identify.

3 A Robust Shallow Parser for Swedish

Most parsers for Swedish are surface oriented, and two of them identify constituency structure. One uses machine learning (Megyesi 02) while the other is based on finite-state cascades (Kokkinakis & Johansson-Kokkinakis 99). Furthermore, two other parsers identify dependency structure using Constraint Grammar (Birn 98) and Functional Dependency Grammar (Voutilainen 01). There is also a deep parser developed in the CLE framework (Gambäck 97). We recently developed

a rule-based shallow parser called Granska Text Analyzer (GTA) (Knutsson *et al.* 03) and due to availability, GTA was used in the experiments. Unfortunately, none of the Swedish parsers could serve as a comparison, since none of them used a comparable constituency structure to that of GTA.

GTA is rule-based and relies on hand-crafted rules written in a context-free formalism. The parser selects grammar rules top-down and uses a passive chart. The rules in the grammar are applied on PoS tagged text, either from an integrated tagger or from an external source. GTA identifies constituents and assigns phrase labels. However, no full trees with a top node are built.

The analysis is surface-oriented and identifies many types of phrases in Swedish. The basic phrase types are adverbial phrases (ADVP), adjective phrases (AP), infinitival verb phrases (INFP), noun phrases (NP), prepositional phrases (PP) and verb phrases (limited) and chains (VP and VC). The internal structure of the phrases is parsed when appropriate and the heads of the phrases are also identified. PP-attachment is left out of the analysis since the parser does not include a mechanism for resolving PP-attachments. The disambiguation of phrase boundaries is primarily done within the rules, and secondly using heuristic selection and disambiguation rules.

In addition to the parsing of phrase structure, clause boundaries (CLB) are detected, resembling Ejerhed's algorithm for clause boundary detection (Ejerhed 99).

The parser was designed for robustness against ill-formed and fragmentary sentences. For example, agreement is not considered in noun phrases and predicative constructions (Swedish has a constraint on agreement in these constructions). By avoiding the constraint for agreement, the parser will not fail due to textual errors or tagging errors. Tagging errors that do not concern agreement are to some extent handled using a set of tag correction rules based on heuristics on common tagging errors.

4 Experiments

We used the toolbox described in Section 2 to evaluate the rule-based parser for Swedish. We used the Stockholm-Umeå corpus (SUC) (Ejerhed *et al.* 92) and chose six random texts (aa02, ac04, je01, jg03, kk03 and kk09) from three different categories, totally amounting to 15 000 words. The text categories were press articles (a), scientific journals (j) and imaginative prose (k). The part-of-speech tag set contained 149 tags. As there exists no constituency tree-bank for Swedish at present, the texts were annotated for tree structure. The texts were first run through the parser and then carefully corrected by a human annotator. The tokenization and sentence boundaries was determined by the corpus.

Since the performance of the parser depends heavily on the performance of the part-of-speech tagger, we compared tagged text from four different sources: the original corpus tags, a hidden Markov model (HMM) tagger, a transformation-based tagger and a baseline tagger. The tagger CORPUS used the original annotations in the SUC corpus, which we assume to have 100% accuracy. The HMM tagger used was TnT (Brants 00), hereafter denoted TNT. The transformation-based tagger (Brill 92) used was fnTBL (Ngai & Florian 01), denoted BRILL. The baseline tagger called BASE chose the most frequent tag for a given word and, for unknown words, the most frequent tag for open word classes. All taggers were trained on SUC data not included in the tests.

To determine the difficulty of the chosen texts, we constructed a baseline parser. To this end, we adopted the approach provided by the CoNLL chunking competition (Tjong Kim Sang & Buchholz 00), i.e. for a given part-of-speech tag, the parse chosen was the most frequent parse for that tag. Given a PoS tagged text, the data was divided into ten parts. Each part was parsed by using the original annotation for the other nine parts as training data. Furthermore, to determine the difficulty of the clause boundary identification we devised a baseline clause identifier simply by assigning `CLB` to the first word of each sentence and `CLI` to the other words.

Thus, we had four taggers (BASE, BRILL, TNT and CORPUS) and two parsers (GTA and baseline). For each combination of tagger and parser, we ran an n-iteration test (using $n = 10$) at each error level (0%, 1%, 2%, 5%, 10% and 20%). In each test, we extracted information about tagging accuracy, parsing accuracy, clause boundary identification and phrase identification for the individual phrase categories ADVP, AP,

Tagger	0%	1%	2%	5%	10%	20%
BASE	85.2	84.4 (0.9)	83.5 (1.9)	81.2 (4.6)	77.1 (9.5)	69.0 (19.0)
BRILL	94.5	93.8 (0.7)	93.0 (1.5)	90.9 (3.8)	87.4 (7.5)	80.1 (15.2)
TNT	95.5	95.0 (0.5)	94.3 (1.2)	92.4 (3.2)	89.5 (6.2)	83.3 (12.7)

Table 1: *Accuracy in percent from the tagging task. The* CORPUS *tagger had 100% accuracy. The columns correspond to the percentage of errors introduced. Relative accuracy degradation compared to the 0% error level is given in brackets.*

Tagger	0%	1%	2%	5%	10%	20%
BASE	81.0	80.2 (0.9)	79.1 (2.3)	76.5 (5.5)	72.4 (10.6)	64.5 (20.3)
BRILL	86.2	85.4 (0.9)	84.5 (1.9)	82.0 (4.8)	78.0 (9.5)	70.3 (18.4)
TNT	88.7	88.0 (0.7)	87.2 (1.6)	85.2 (3.9)	81.7 (7.8)	75.1 (15.3)

Table 2: *Accuracy in percent from the parsing task. Parsing based on the* CORPUS *tagger had 88.4% accuracy. A baseline parser using the* CORPUS *tagger had 59.0% accuracy.*

INFP, NP, PP and VC. Also, since some tokens are outside all phrases, we included an outside category (O).

5 Results

The most important aspect of the accuracy of the GTA parser is the performance of the underlying tagger. Most taggers were quite robust against ill-formed and noisy input as seen from Table 1. For example, at the 20% error level, TNT degraded 12.7% and BRILL degraded 15.2% relatively to their initial accuracy of 95.5% and 94.5%, respectively. The low degradation in performance is most likely due to the robust handling of unknown words in BRILL and TNT, where the suffix determines much of the morphological information. Thus, if the last letters of a word are unaffected by a spelling error, the tag is likely to remain unchanged. The robustness of the baseline tagger was not as satisfactory as it guessed the wrong tag in almost all cases (19.0% of 20%). The baseline tagging accuracy for text without errors was 85.2%.

For the parsing task, we obtained 86.2% accuracy using BRILL and 88.7% accuracy using TNT, as seen in Table 2. An interesting observation is that the accuracy of parsing using CORPUS, i.e. perfect tagging, was 88.4%, which is lower than that of TNT. The explanation is found in the way the taggers based on statistics generalize from the training data. The CORPUS tagger adopts the noise from the manual annotation of the SUC corpus, which will make the task harder for the parser. This is further substantiated below when we discuss the baseline parser.

The degradation at the 20% error level seems promising since the accuracy only dropped 15.3% using the TNT tagger. On the other hand, since the performance of TNT had already degraded 12.7% in tagging accuracy, the additional $15.3 - 12.7 = 2.6\%$ was due to the fact that the context surrounding a tagging error was erroneously parsed. This difference is the degradation of the parser in isolation. Nevertheless, the performance of the whole system is the most relevant measure, since the most accurate tagger does not necessarily provide the best input to the rest of the parsing system. As stated earlier, since this paper describes evaluation methodology only, we do not address how the errors affected the syntactic structure.

As a comparison, the baseline parser using the CORPUS tagger had 59.0% accuracy, while the TNT tagger obtained 59.2%. This further indicates that the difference between TNT and CORPUS is real and not just an idiosyncrasy of the parsing system. A system not using any knowledge at all, i.e. the baseline parser using the BASE tagger, obtained 55.5% accuracy.

As seen from Table 3, the task of clause identification (`CLB`) was more robust to ill-formed input than any other task with only 7.0% degradation using TNT at the 20% error level. This may be attributed to the fact that half the clause delimiters resided at the beginning of a sentence and thus, were unaffected by spelling errors. Of course, the

Tagger	0%	1%	2%	5%	10%	20%
BASE	84.2	84.0 (0.2)	83.6 (0.7)	82.9 (1.5)	81.9 (2.7)	79.4 (5.7)
BRILL	87.3	87.0 (0.3)	86.6 (0.8)	85.6 (1.9)	83.8 (4.0)	80.3 (8.0)
TNT	88.3	87.9 (0.4)	87.5 (0.9)	86.6 (1.9)	85.1 (3.6)	82.1 (7.0)

Table 3: *F-score from the clause boundary identification task. Identification based on the* CORPUS *tagger had an F-score of 88.2%. A baseline identifier had an F-score of 69.0%. The columns correspond to the percentage of errors introduced. Relative accuracy degradation compared to the 0% error level is given in brackets.*

Type	0%	1%	2%	5%	10%	20%	Count
ADVP	81.9	81.3 (0.7)	80.6 (1.5)	78.6 (4.0)	75.3 (8.0)	68.4 (16.4)	1008
AP	91.3	90.5 (0.8)	89.8 (1.6)	87.0 (4.7)	83.1 (8.9)	74.3 (18.6)	1332
INFP	81.9	81.4 (0.6)	80.9 (1.2)	79.2 (3.2)	76.0 (7.2)	70.2 (14.2)	512
NP	91.4	90.9 (0.5)	90.2 (1.3)	88.4 (3.2)	85.2 (6.7)	79.3 (13.2)	6895
O	94.4	94.2 (0.2)	93.9 (0.5)	93.3 (1.1)	92.1 (2.4)	89.9 (4.7)	2449
PP	95.3	94.8 (0.5)	94.3 (1.0)	93.0 (2.4)	90.9 (4.6)	85.8 (9.9)	3886
VC	92.9	92.3 (0.6)	91.5 (1.5)	89.8 (3.3)	86.8 (6.5)	80.9 (12.9)	2562
Total	88.7	88.0 (0.7)	87.2 (1.6)	85.2 (3.9)	81.7 (7.8)	75.1 (15.3)	

Table 4: *F-scores for the individual phrase categories from the parse task.* TNT *was used to tag the text.*

baseline clause identifier was also unaffected by spelling errors and obtained a 69.0% F-score for all error levels. Clause identification at 0% error level achieved an 88.3% F-score (88.3% recall, 88.3% precision) using TNT.

We provide the F-scores for the individual phrase categories using TNT in Table 4. We see that adverbial (ADVP) and infinitival verb phrases (INFP) are much less accurate than others. They are also among the most sensitive to ill-formed input. In the case of INFP, this may be attributed to the fact that they are often quite long and an error introduced near or at the infinite marker or the verb is detrimental. In the count column, we provide the number of rows in which a given phrase type occurs in the annotation. For example, in the case of NP, we count the number of rows in which at least one `NPB` or `NPI` occurs.

Standard deviation was calculated for all accuracy and F-score values at each error level, by using data from the n runs. Standard deviations were low for all tasks and were 0.13, 0.22 and 0.22 on the average for Tables 1, 2 and 3, respectively. The maximum standard deviation was 0.70 for the 20% error level for clause boundary identification using TNT. Standard deviation was 0.49 on the average for Table 4. The only noticeable exception was the infinitival verb phrase (INFP), which had a 2.5 standard deviation at the 20% error level using the BRILL tagger.

Note that 15 000 words may not be sufficient for a reliable conclusion on robustness. The experiments here are primarily provided to illustrate the evaluation method. The results from the evaluation are based on non-tuned output from the parser compared to the manually annotated data. Furthermore, there are still some minor inconsistencies between parser output and the annotation scheme. This is of course a source of systematic errors and will be dealt with in a near future.

6 Conclusions

We have described an automatic framework for testing the robustness of tagging and parsing. We introduced spelling errors resulting in non-existing words in order to feed the parsing system with ill-formed and noisy data. From this, the different components of a parsing system can be individually evaluated. With increasing levels of noise, the performance of the system will inevitably degrade. Here, we have addressed the difference between tagging the original, error-free text and tagging noisy text and found that state-of-the-art tagging is quite robust against ill-formed input. Furthermore, we have discussed the effect of tagging performance degradation on the over-all performance of parsing systems. Experi-

ments conducted on a shallow parser for Swedish exhibited graceful degradation in over-all parsing performance. To conclude, we advocate the use of the proposed automatic evaluation method to obtain fair and reliable measures of over-all parsing system performance, as well as a measure of performance of the individual components.

References

(Bigert *et al.* 03) J. Bigert, L. Ericson, and A. Solis. Missplel and AutoEval: Two generic tools for automatic evaluation. In *Proceedings of Nodalida 2003*, Reykjavik, Iceland, 2003.

(Birn 98) J. Birn. Swedish constraint grammar. Technical report, Lingsoft Inc, Helsinki, Finland, 1998.

(Brants 00) T. Brants. TnT – a statistical part-of-speech tagger. In *Proceedings of ANLP-2000*, Seattle, USA, 2000.

(Brill 92) E. Brill. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92*, pages 152–155, Trento, Italy, 1992.

(Carroll *et al.* 98) J. Carroll, T. Briscoe, and A. Sanfilippo. Parser evaluation: a survey and a new proposal. In *Proceedings of LREC 1998*, pages 447–454. Granada, Spain, 1998.

(Damerau 64) F. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

(Ejerhed 99) E. Ejerhed. Finite state segmentation of discourse into clauses. In A. Kornai, editor, *Extended Finite State Models of Language*, chapter 13. Cambridge University Press, 1999.

(Ejerhed *et al.* 92) E. Ejerhed, G. Källgren, O. Wennstedt, and M. Åström. *The Linguistic Annotation System of the Stockholm-Umeå Project*. Department of Linguistics, University of Umeå, Sweden, 1992.

(Gambäck 97) B. Gambäck. *Processing Swedish Sentences: A Unification-Based Grammar and some Applications*. Unpublished PhD thesis, The Royal Institute of Technology and Stockholm University, 1997.

(Knutsson *et al.* 03) O. Knutsson, J. Bigert, and V. Kann. A robust shallow parser for Swedish. In *Proceedings of Nodalida 2003*, Reykjavik, Iceland, 2003.

(Kokkinakis & Johansson-Kokkinakis 99) D. Kokkinakis and S. Johansson-Kokkinakis. A cascaded finite-state parser for syntactic analysis of Swedish. In *Proceedings of the 9th EACL*, pages 245–248, Bergen, Norway, 1999. Association for Computational Linguistics.

(Li & Roth 01) X. Li and D. Roth. Exploring evidence for shallow parsing. In W. Daelemans and R. Zajac, editors, *Proceedings of CoNLL-2001*, pages 38–44, Toulouse, France, 2001.

(Megyesi 02) B. Megyesi. Shallow parsing with PoS taggers and linguistic features. *Journal of Machine Learning Research*, Special Issue on Shallow Parsing(2):639–668, 2002.

(Menzel 95) W. Menzel. Robust processing of natural language. In *Proceedings of 19th Annual German Conference on Artificial Intelligence*, pages 19–34, Berlin, Germany, 1995.

(Ngai & Florian 01) G. Ngai and R. Florian. Transformation-based learning in the fast lane. In *Proceedings of NAACL-2001*, pages 40–47, Carnegie Mellon University, Pittsburgh, USA, 2001.

(Nivre 02) J. Nivre. What kinds of trees grow in Swedish soil? a comparison of four annotation schemes for Swedish. In *Proceedings of First Workshop on Treebanks and Linguistic Theories (TLT2002)*, pages 123–138, Sozopol, Bulgaria, 2002.

(Ramshaw & Marcus 95) L. Ramshaw and M. Marcus. Text chunking using transformation-based learning. In D. Yarovsky and K. Church, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94, Somerset, New Jersey, 1995.

(Tjong Kim Sang & Buchholz 00) E. Tjong Kim Sang and S. Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132. Lisbon, Portugal, 2000.

(Voutilainen 01) A. Voutilainen. Parsing Swedish. In *Proc. of Nodalida 01 - 13th Nordic Conference on Computational Linguistics*, 2001.