Error Detection for Treebank Validation

Bharat Ram Ambati
LTRC, IIIT-Hyderabad
ambati@research.iiit.ac.in

Rahul Agarwal
LTRC, IIIT-Hyderabad
rahul.agarwal@research.iiit.ac.in

Mridul Gupta
LTRC, IIIT-Hyderabad
mridulgp@gmail.com

Samar Husain
LTRC, IIIT-Hyderabad
samar@research.iiit.ac.in

Dipti Misra Sharma
LTRC, IIIT-Hyderabad
dipti@iiit.ac.in

Abstract

This paper describes an error detection mechanism which helps in validation of dependency treebank annotation. Consistency in treebank annotation is a must for making data as error-free as possible and for assuring the usefulness of treebank. This work is aimed at ensuring this consistency and to make the task of validation cost effective by detecting major errors induced during completely manual annotation. We evaluated our system on the Hindi dependency treebank which is currently under development. We could detect 76.63% of errors at dependency level. Results show that our system performs well even when the training data is low.

1 Introduction

For effective processing of text, tools at different conceptual levels, say from letter/syllable level to discourse level are needed. Output of these tools can then be used in different NLP based applications, beginning with simple spell checkers to sophisticated machine translation systems. These tools could be completely rule-based, statistical or hybrid systems. To build such tools, manually annotated gold standard corpora are required. Annotated corpora are mostly obtained by either manual or semi-automated processes. Hence, there are chances that errors are introduced either by human annotators or by the pre-processed output of automated tools. It is crucial that the annotated corpora are free of anomalies (errors) and inconsistencies. In the process of making these corpora error free, experts need to validate them. As the data is already annotated carefully (which is a time-consuming task), we need tools that can supplement the validators' task with a view of making the overall task fast, without compromising on reliability. With the help of such a tool, a validator can directly go to error instances and correct them. Therefore, we need the tool to have high recall. It is easy to see that a human validator can reject unintuitive errors (false positives) without much effort; one can therefore compromise a little bit on precision.

In this paper, we propose an error detection mechanism to detect dependency errors in the Hindi treebank (Bhatt et al., 2009) annotation. We classify the identified errors under specific categories for the benefit of the validators, who may choose to correct a specific type of error at one time. Though we did experiments on Hindi treebank, our approach can be applied to any under developing treebank with minimal effort.

The paper is arranged as follows. Section 2 gives a brief overview of the Hindi dependency treebank. Details of the information annotated, annotation procedure and types of possible errors in the treebank are discussed in section 3. We present the related work in section 4. In section 5, we describe our approach. Results of our approach are presented in section 6. Section 7 focuses on a general discussion about the results and approach and proposes a future direction to our work. We conclude our paper in section 8.

23

2 Hindi Dependency Treebank

A multi-layered and multi-representational treebank for Hindi (Bhatt et al., 2009; Xia et al., 2009) is being developed. The treebank will have dependency, verb-argument (PropBank, Palmer et al., 2005) and phrase structure (PS) representation. Automatic conversion from dependency structure (DS) to phrase structure (PS) is planned. Hence, it is important to have a high quality version of the dependency treebank so that the process of automated conversion to PS does not induce errors in PS. The dependency treebank contains information encoded at the morpho-syntactic (morphological, part-of-speech and chunk information) and syntactico-semantic (dependency) levels.

3 Dependency Representation

In this section we first describe the information encoded in the dependency representation of the treebank. We then briefly describe the annotation procedure for encoding this information. We also describe the possible errors at each level of annotation.

3.1 Information encoded in dependency representation

During dependency annotation, Part-Of-speech (POS), morph, chunk and inter-chunk dependency relations are annotated. Some special features are also annotated for some specific nodes. Details can be seen in this section.

Part-Of-Speech (POS) Information: POS tags are annotated for each node following the POS and chunk annotation guidelines (Bharati et al., 2006).

Morph Information: Information pertaining to the morphological features of the nodes is also encoded using the Shakti standard format (SSF) (refer, Bharati et al., 2007). These morphological features have eight mandatory feature attributes for each node. These features are classified as root, category, gender, number, person, case, post position (for a noun) or tense aspect modality (for a verb) and suffix.

Chunk Information: After annotation of POS tags, chunk boundaries are marked with appropriate assignment of chunk labels (Bharati et al., 2006). This information is also stored in SSF (Bharati et al., 2007).

Dependency Relations: After POS, morph and chunk annotation, inter-chunk dependency annotation [1] is done following the set of dependency guidelines in Bharati et al. (2009). This information is encoded at the syntactico-semantic level following the Paninian dependency framework (Begum et al., 2008; Bharati et al., 1995). After inter-chunk annotation, plan is to use a high accuracy intra-chunk expander, which marks the intra-chunk dependencies [2] and expands the chunks arriving at sentence level dependency tree.

Other Features: In the dependency treebank, apart from POS, morph, chunk and inter-chunk dependency annotations, some special features for some specific nodes are marked. For example, for the main verb of a sentential clause, information about whether the clause is declarative, interrogative or imperative is marked. Similarly, whether the sentence is in active or passive voice is also marked.

3.2 Annotation Procedure

At POS, chunk and morphological levels, corresponding state-of-the-art tools are run as a first step. An annotator then checks each node and corrects the tool's output. Another annotator (validator) takes the annotated data and validates it. At dependency level, due to unavailability of high accuracy parser, manual annotation followed by validation is done.

Annotation is being done using a tool called Sanchay[3]. Sanchay is an open source platform for working on languages, especially South Asian languages, using computers and also for developing Natural Language Processing (NLP) or other text processing applications. Apart from syntactic annotation interface (used for Hindi dependency annotation), it has several other useful functionalities as well. Font conversion, language and encoding detection, n-gram generation are a few of them.

3.3 Types of possible errors at various levels

In this section we describe the types of annotation errors at each level and provide examples for some specific types of errors.

---

[1] Inter-chunk dependencies are the dependency relations marked between the chunks, chunk heads to be specific.
[2] Intra-chunk dependencies are the dependency relations marked with in the chunk.
[3] http://sanchay.co.in/.

POS Errors: In POS errors we try to identify whether the Part-Of-Speech (POS) tag is correct or not for each lexical item. For example, in the example sentence given below '*chalaa*' should be the main verb (*VM*) instead of an auxiliary verb (*VAUX*).

raama	ghara	chalaa	gayaa.
NNP	NN	VAUX	VAUX
'Ram'	*'home'*	*'walk'*	*'went'*.

"Ram went home".

Morph Errors: Errors in the eight attribute values as mentioned in the previous section are classified as morph errors.

Chunk Errors: There can be two types of chunk errors. One is chunk type and the other is chunk boundary. In chunk type we identify whether the chunk label is correct or not. In chunk boundary we identify whether the node should belong to the same chunk or different chunk. For example, consider the following chunk,

((		NP
meraa	*'my'*	PRP
bhaaii	*'brother'*	NN
))		

In Hindi, '*meraa*' and '*bhaaii*' should be in two separate noun chunks (refer Bharati et al., 2006). So, in the above example, the chunk label of '*bhaaii*' is correct, but the boundary is wrong.

Dependency Errors: In dependency errors we try to identify whether a node is attached to its correct parent and whether its dependency label is correct or not. In addition to dependency relation errors, we also identify errors in general linguistic constraints and framework specific errors, for example, the tree well-formedness assumption in dependency analysis. Framework specific example would be that children of a conjunct should be of similar type (Bharati et al., 2009). For example, a conjunct can have two nouns as its children but not a noun and a verb as its children.

Other Feature Errors: Errors in the special features discussed above are classified under other feature errors.

Focus of the current paper is to describe the methodology employed to detect errors in the dependency level (inter-chunk dependencies) of the DS representation. Error detection at intra-chunk dependencies is out of scope of this paper. In the rest of the paper, by dependency level, we mean inter-chunk dependencies unless explicitly stated. In the following section, we first describe the related work in the area of detecting errors in treebanks, in general. Then, we present the work on Hindi in particular.

4 Related Work:

Validation and correction tools are an important part for making treebanks error-free and consistent. With an increase in demand for high quality annotated corpora over the last decade, major research works in the field of developing lexical resources have focused on detection of errors.

One such approach for treebank error detection has been employed by Dickinson and Meurers (2003a; 2003b; 2005) and Boyd et al. (2008). The underlying principle in these works is to detect "variation n-grams" in syntactic annotation across large corpora. These variations could be present for a continuous sequence of words (POS and chunks) or for a non-continuous sequence of words (dependency structures), more the number of variation for a particular contiguous or non-contiguous sequence of tokens (or words), greater the chance of the particular variation being an error. They use these statistical patterns (n-grams) to detect anomalies in POS annotation in corpora such as the Penn treebank (Marcus et al., 1993), TIGER corpus (Brants et al., 2002) etc. For discontinuous patterns as found most commonly in dependency annotation (Boyd et al., 2008), they tested their strategy on Talbanken05 (Nivre et al., 2006) apart from the corpora mentioned above. This we believe was the first mainstream work on error detection in dependency annotation.

Some other earlier methods employed for error detection in syntactic annotation (mainly POS and chunk), are by Eskin (2000) and van Halteren (2000). Based on large corpora, van Noord (2004) and de Kok et al. (2009) employed error mining techniques. The basic underlying strategy was to obtain a set of parsed and un-parsed sentences using a wide-coverage parser and compute suspicion ratio for detecting errors. Other examples of detection of annotation errors in treebanks include Kaljurand (2004) and Kordoni (2003).

Most of the aforementioned techniques work well with large corpora in which the frequency of occurrence of words is very high. Hence, none of them account for data sparsity except for de Kok et al. (2009). Moreover, the techniques employed

by van Noord (2004) and de Kok et al. (2009) rely on the output of a reliable state-of-the-art parser which may not be available for many languages just as in the case of Hindi, the language in question for our work.

It becomes a challenge to develop error detection tools for small treebanks like Hindi. There is an effort by Ambati et al. (2010) in this direction for Hindi treebank validation. They used a combination of a rule-based and hybrid system to detect treebank errors. Rule-based system works on the development of robust (high precision) rules which are formed using the annotation guidelines and the framework, whereas the hybrid system is a combination of statistical module with a rule-based post-processing module. The statistical module helps in detecting a wide array of potential errors and suspect cases. The rule-based post-processing module then prunes out the false positives, with the help of robust and efficient rules thereby ensuring higher precision value.

Note that both "Rule-Based Approach" and "Rule-based post-processing" modules have separate goals. Goal of "Rule-Based Approach" is to detect errors using high precision rules. Whereas goal for the later, is to prune the false positives given by the statistical approach. Former one tries to increase the precision of the system, whereas the later tries to increase the recall of the system. Rules used in "Rule-Based Approach" can be used in post-processing module, but vice versa is not true. The entire framework is sketched in Figure 1.
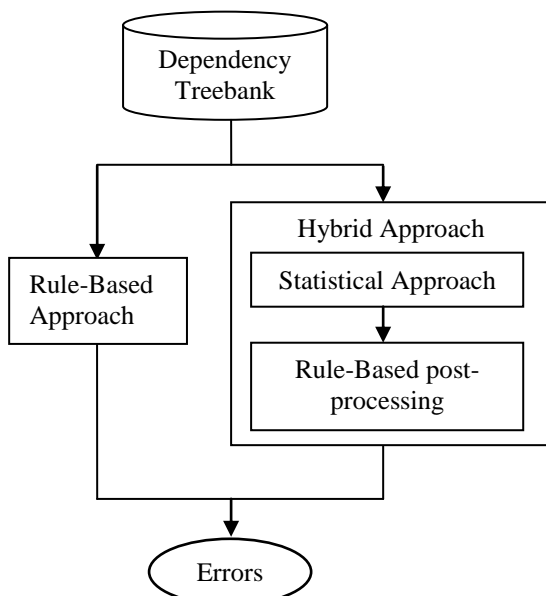

Figure 1. Error detection framework by Ambati et al. (2010)

Ambati et al. (2010) could detect errors in POS and chunk annotations with reasonable accuracies. But at dependency level recall of overall system (combination of rule-based and hybrid approaches) is 40.33% only. This is mainly due to low performance of the statistical module. In statistical module, they extracted frequencies of child and parent node bi-grams in the dependency tree. To handle scarcity issues, they explored different similarity measures and merged similar patterns. We call this as "Frequency Based Statistical Module (FBSM)". Major limitation of this approach is that one cannot give richer context due to the problem of scarcity. To find whether the dependency label is correct or not, apart from node and its parent information, contextual features like sibling and child information is also helpful. Current state-of-the-art dependency parsers like MSTParser[4] and MaltParser[5] use these features for dependency labeling (McDonald et al., 2006; Nivre et al., 2007; Kosaraju et al., 2010). Finding similarity between patterns and merging similar patterns would not help when we wish to take a much richer context.

In this paper, we propose a probability based statistical module (PBSM) to overcome this problem of FBSM. With this approach, we evaluate and compare the performance of PBSM and FBSM. Now in place of FBSM, we integrate PBSM into overall system of Ambati et al. (2010) and compare the results.

5   Our Approach: Probability Based Statistical Module (PBSM):

In this probability based statistical module, we first extract the contextual features which help in identifying the correct tag. For example, at the dependency level, apart from node and its parent features, sibling and child features with their respective dependency labels are very useful in predicting the correct dependency label. Using these contextual features from the training data, we create a model using maximum entropy classification algorithm[6] (MAXENT). This model gives the probabilities of all possible output tags (here dependency labels) for a given context. For each node in the test data, we first extract the context information and the input tag of that node. We then extract the list of all possible dependency tags with their probabilities for this

---

[4] http://sourceforge.net/projects/mstparser/
[5] http://maltparser.org/
[6] http://maxent.sourceforge.net/

context using the trained model. From this list we extract first best and second best tags and their corresponding probabilities.

If the input tag doesn't match with the first best tag, and if the probability of the first best tag is greater than a particular threshold, we then consider it as a possible error node. These could be valid errors or the cases which require much richer context to find the correct tag.

If both the input tag and the first best tag given by the model match, we then fix a maximum and minimum threshold on the probability values. If the probability of the first best tag is greater than the maximum threshold, we do not consider it as a potential error. The chance of it being an error is very low as the system is very confident about its decision. If the probability of the first best tag is less than the minimum threshold, it is considered as a possible error. This could either be the case of an error pattern or a correct but less frequent pattern. If it is the latter, then the rule-based post-processing tool will remove this false positive.

If the probability value lies between the maximum and minimum thresholds, we calculate the difference between the probabilities of the first and second best tags. If the difference is less than a particular value, it means that there is high ambiguity between these two tags. As there is high ambiguity there is a greater chance of making an error. Hence, we identify this case as a possible error. In this way using the probabilistic approach, we not only detect the possible errors, but also classify them into different categories.

find_errors (input)

```
    for each sentence in the input:
      for each node in the sentence:
1.      Get the node's context
2.      Get the input_tag
3.      tags_probs = get_probabilities(context);
4.      1stBestTag = tags_probs[0][0];
5.      1stBestProb = tags_probs[0][1];
6.      2ndBestTag = tags_probs[1][0];
7.      2ndBestProb = tags_probs[1][1];
8.      if input_tag != 1stBestTag:
          if 1stBestProb > thres_minX:
            mark node as error node (less context);
9.      else:
          if 1stBestProb < thres_max:
            if 1stBestProb < thres_min:
              mark node as error node (less frequent);
            else if 1stBestProb-2ndBestProb < thres_dif:
              mark node as error node (ambiguous);
```

get_probabilities(context)

```
1.  Load the trained maxent model
2.  Predict probabilities of all tags for the context.
3.  Store the tags and their probabilities in an array.
4.  Return the array.
```

Figure 2. Algorithm employed for PBSM

Figure 2 shows the algorithm of probability based statistical module (PBSM). Use of richer contextual information and probabilities to detect errors makes this approach more effective from the previous approaches employed for error detection (Dickinson and Meurers, 2003a; 2003b; 2005; Boyd et al., 2008; Ambati et al., 2010).

Using this approach, not only can one detect errors but also classify them under specific categories like less context, less frequent and ambiguous cases, which will help the validation process. This helps the validators to correct the errors in a focused way. For example, a validator can check and correct all the error in "less fre-

quent" category first and then start correcting "ambiguous cases". It also helps validator to decide the amount of energy he/she needs to spend. For example, correcting "ambiguous cases" would require more time compared to other categories. This could be because the validator might look for sentence or sometimes discourse level information to resolve the ambiguity. He/she could also contact peers or an expert to resolve it. Hence, more time might be required to resolve "ambiguous cases" compared to others.

6 Experiments and Results

We used same data used by Ambati et al. (2010) for evaluation. This is a 65k-token manually annotated and validated sample of data (2694 sentences) derived from the Hindi dependency treebank. The data is divided into 40k, 10k and 15k for training, development and testing respectively. We used training data to train the model and development data to tune the parameters like threshold values. For our experiments, *thres_max*= 0.8, *thres_min* = 0.2, *thres_minX* = 0.25 and *thres_dif* = 0.25 gave the best performance.

Table 1 shows the performance of PBSM and compares it with FBSM. FBSM of Ambati et al. (2010) could identify only 18.74% of the dependency errors. The precision recorded for this approach was also quite low. But with our PBSM, we could detect 57.06% of the dependency errors with a reasonable precision value. Note that, our main aim is to achieve a high recall value. The false positives can be easily discarded by the validators.

Approach	Total Errors (Total instances)	System output	Correct Errors	Recall
FBSM of Ambati et al. (2010)	843 (7113)	2546	158	18.74%
PBSM: Our Approach	843 (7113)	2000	481	57.06%

Table 1. Error detection at dependency level using FBSM of Ambati et al. (2010) and our PBSM

Overall system recall of Ambati et al. (2010) for error detection at dependency level is 40.33%. This system uses FBSM in hybrid approach. We replaced FBSM with our PBSM and re-evaluated the overall system of Ambati et al.

(2010). The modified system could achieve a recall of 76.63% with reasonable precision of 29.84%. Results are shown in Table 2.

Approach	Total Errors	System output	Correct Errors	Recall
Ambati et al. (2010) overall system with FBSM	843	2728	340	40.33%
Ambati et al. (2010) overall system with PBSM	843	2165	646	76.63%

Table 2. Error Detection at dependency level using overall system of Ambati et al. (2010) with FBSM and PBSM

7 Discussion and Future work

Proposed PBSM identifies 38% more errors than the FBSM at dependency level. As we have less data, hypothesis for FBSM, "Low frequency is a possible sign of error", didn't work. Unsurprisingly, several valid patterns had low counts. Major advantage of PBSM over FBSM is the use of richer context. Richer context helped PBSM to predict the errors more accurately. But in FBSM we couldn't use it because of sparsity issues. Results show that our approach works well even when the size of the data is low.

The tool is being constantly improved. We are analyzing the errors which are missed out and planning to improve. Currently, precision of the system is low. Improving the "Rule-based post-processing" step of hybrid approach can significantly increase the precision. We can build an interface where a validator while validating the data can automatically add new rules to this post-processing step. We would also like to evaluate our system on the time taken for validation. That is the reduction in the validation time using this system. We are also planning to build a user-friendly interface which helps validators in correcting the errors.

This system can also help in improving the guidelines which subsequently improves the annotation. While correcting the errors if the validator comes across some ambiguous decisions or some common errors or comes up with new decisions, guidelines can be modified accordingly to reflect the changes. Data annotated based on new guidelines will reduce the occurrence of these errors and eventually the quality of annotation of individual as well as entire data will improve.

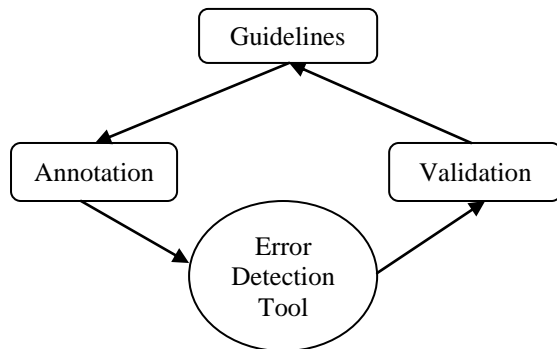Figure 3, shows the complete cycle of this process.

Figure 3. Cycle for improving guidelines for annotation

Although we worked and presented our results only on the Hindi Treebank, our approach can be generalized to any language and to any framework. Parameter tuning like the threshold values is the only part which depend on the size of data, language and the framework.

8 Conclusions

We proposed a novel approach to detect errors in the treebanks. This approach can significantly reduce the validation time. We tested it on Hindi dependency treebank data and were able to detect 76.63% of errors at dependency level. This tool can be generalized to detect errors in annotation of any language/framework. Results show that the proposed approach works well even when the size of the data is low.

References

B. R. Ambati, M. Gupta, S. Husain and D. M. Sharma. 2010. A high recall error identification tool for Hindi treebank validation. In *Proceedings of The 7th International Conference on Language Resources and Evaluation (LREC), Valleta, Malta.*

R. Begum, S. Husain, A. Dhwaj, D. M. Sharma, L. Bai, and R. Sangal. 2008. Dependency annotation scheme for Indian languages. In *Proceedings of IJCNLP-2008.*

A. Bharati, V. Chaitanya and R. Sangal. 1995. *Natural Language Processing: A Paninian Perspective, Prentice-Hall of India, New Delhi,* pp. 65-106.

A. Bharati, R. Sangal, D. M. Sharma and L. Bai. 2006. AnnCorra: Annotating Corpora Guidelines for POS and Chunk Annotation for Indian Languages. *Technical Report (TR-LTRC-31), Language Technologies Research Centre, IIIT-Hyderabad.*

A. Bharati, R. Sangal and D. M. Sharma. 2007. SSF: Shakti Standard Format Guide. *Technical Report (TR-LTRC-33), LTRC, IIIT-Hyderabad.*

A. Bharati, D. M. Sharma S. Husain, L. Bai, R. Begam and R. Sangal. 2009. AnnCorra: TreeBanks for Indian Languages, Guidelines for Annotating Hindi TreeBank (version – 2.0). http://ltrc.iiit.ac.in/MachineTrans/research/tb/DS-guidelines/DS-guidelines-ver2-28-05-09.pdf

R. Bhatt, B. Narasimhan, M. Palmer, O. Rambow, D. M. Sharma and F. Xia. 2009. Multi-Representational and Multi-Layered Treebank for Hindi/Urdu. In *Proc. of the Third Linguistic Annotation Workshop at 47th ACL and 4th IJCNLP.*

A. Boyd, M. Dickinson, and W. D. Meurers. 2008. On Detecting Errors in Dependency Treebanks. *Research on Language and Computation* 6(2), pp. 113-137.

S. Brants, S. Dipper, S. Hansen, W. Lezius and G. Smith, 2002. The TIGER Treebank. In *Proceedings of TLT-02.* Sozopol, Bulgaria.

M. Dickinson and W. D. Meurers. 2003a. Detecting Inconsistencies in Treebank. In *Proc. of the Second Workshop on Treebanks and Linguistic Theories (TLT 2003).*

M. Dickinson and W. D. Meurers. 2003b. Detecting Errors in Part-of-Speech Annotation. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL-03).* Budapest, pp. 107–114.

M. Dickinson and W. D. Meurers. 2005. Detecting Errors in Discontinuous Structural Annotation. In *Proc. of the 43rd Annual Meeting of the ACL*, pp. 322–329.

E. Eskin. 2000. Automatic Corpus Correction with Anomaly Detection. In *Proceedings of the First Conference of the North American Chapter of the-Association for Computational Linguistics (NAACL-00).* Seattle, Washington.

Hans van Halteren. 2000. The Detection of Inconsistency in Manually Tagged Text. In *Proceedings of the 2ndWorkshop on Linguistically Interpreted Corpora*. Luxembourg.

K. Kaljurand. 2004. Checking treebank consistency to find annotation errors. http://math.ut.ee/˜kaarel/NLP/Programs/Treebank/ConsistencyChecking/

Daniel de Kok, Jianqiang Ma and Gertjan van Noord. 2009. A generalized method for iterative error mining in parsing results. In *Proceedings of Workshop on Grammar Engineering Across Frameworks (GEAF 2009), 47th ACL – 4th IJCNLP*, Singapore.

V. Kordoni. 2003. Strategies for annotation of large corpora of multilingual spontaneous speech data. In *Proc. of Workshop on Multilingual Corpora: Lin-*

guistic Requirements and Technical Perspectives held at Corpus Linguistics 2003.

P. Kosaraju, S. R. Kesidi, V. B. R. Ainavolu and P. Kukkadapu. 2010. Experiments on Indian Language Dependency Parsing. In *Proceedings of the ICON10 NLP Tools Contest: Indian Language Dependency Parsing*.

M. P. Marcus, M. A. Marcinkiewicz, B. Santorini. 1993. Building a large annotated corpus of English: the Penn treebank. *Computational Linguistics*, *Volume 19*, Issue 2 (313 – 330).

R. McDonald, K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pp. 216–220.

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov and E Marsi. 2007. *Malt-Parser: A language-independent system for data-driven dependency parsing. Natural Language Engineering*, 13(2), 95-135.

J. Nivre, J. Nilsson and J. Hall. 2006. Talbanken05: A Swedish Treebank with Phrase Structure and Dependency Annotation. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC- 06)*. Genoa, Italy.

Gertjan van Noord. 2004. Error Mining for Wide-Coverage Grammar Engineering. In *Proceedings of ACL 2004*, Barcelona, Spain.

M. Palmer, D. Gildea, P. Kingsbury. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, *31(1):*71-106.

F. Xia, O. Rambow, R. Bhatt, M. Palmer, and D. M. Sharma. 2009. Towards a Multi-Representational Treebank. In *Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories (TLT 2009)*, Groningen, Netherlands.