**L**inguistic **I**ssues in **L**anguage **T**echnology – **LiLT**
Submitted, January 2012

# A Hybrid Approach to Error Detection in a Treebank

## Its Impact on Manual Validation Time

**Rahul Agarwal**
**Bharat Ram Ambati**
**Dipti Misra Sharma**

Published by CSLI Publications

**LiLT** volume 7, issue 20 January 2012

# A Hybrid Approach to Error Detection in a Treebank

## Its Impact on Manual Validation Time

Rahul Agarwal , *LTRC, IIIT-Hyderabad*
Bharat Ram Ambati , *LTRC, IIIT-Hyderabad*
Dipti Misra Sharma , *LTRC, IIIT-Hyderabad*

### Abstract

Treebanks are a linguistic **resource**: a large database where the morphological, syntactic and lexical information for each sentence has been explicitly marked. The critical requirements of treebanks for various NLP activities (research and application) are well known. This also implies that treebanks need to be as error free as possible. However, manual validation of a treebank is very costly, both in terms of time and money. This paper describes an approach to automatically detect errors in a treebank after a complete manual annotation. Over and above improving an earlier error detection tool (Ambati et al. (2011)) for a Hindi treebank. We also present a user study to show that our system reduces the validation time significantly while detecting 81.49% of the errors at the dependency level.

**LiLT** Volume 7, Issue 20, January 2012.
*A Hybrid Approach to Error Detection in a Treebank.*

## 1 Introduction

Treebanks have proved to be a crucial resource for NLP research and developing solutions for various NLP related applications. A treebank should be error free considering its role in providing appropriate linguistic knowledge. However, manual detection of errors is a very costly task, both in terms of time and money. Thus, automatic error detection tools are required to reduce the time of validation. The tools could be rule based or statistical or combination of the two. Hence, tools with high recall value and significant precision are highly favored for this task. The time for validation can be further trimmed down using GUIs which can supplement validators as explained by Ambati et al. (2011).

The process of developing a treebank involves manual or semi-automatic annotations of linguistic information. A semi-automatic procedure involves annotating the grammatical information using tools. Output of these tools is then manually checked and corrected. Both these procedures may leave errors in the treebank on the first attempt. Therefore, there is usually another step of identifying and manually correcting these errors.

In this work, we improve over the mechanism proposed by Ambati et al. (2011) to detect dependency annotation errors. The data used for the experiments is a part of the larger Hindi dependency treebank (Bhatt et al. (2009); Xia et al. (2009)). We used this data to show the performance of our system. We integrate our system with Sanchay[1].

For more details on the type of errors which we extract from the Hindi dependency treebank, please refer to our previous work (Ambati et al. (2011)).

The paper is arranged as follows. In section 2, we summarize some of the previous work done on detecting errors in a treebank. In section 3, we briefly describe our approach. Experiments and results are given in section 4. In section 5, we present a user study, performed to check whether our system is able to reduce the overall validation time. In section 6, we describe a few system improvements that reduce the validation time. Section 7 contains a general discussion and concludes our paper.

## 2 Related Work

Error detection in a treebank has gained interest for researchers over the last decade due to an increase in the demand for a high quality annotated corpora.

[1] http://www.sanchay.co.in

Dickinson and Meurers (2003a,b) developed a treebank error detection approach based on the concept of 'ngram variation detection'. In 'ngram variation detection', they try to find strings, which occur multiple times in the corpus, but have varying syntactic annotations. This can be because the strings are ambiguous and can have different structures, depending on the meaning, or because the annotation is erroneous in at least one of the cases. The idea was then adapted for dependency structures (Dickinson and Meurers (2005)) as well, by analyzing the possible dependency relations between same words. Again different dependency relations can be either the result of ambiguity or errors.

Some other earlier methods for error detection in syntactic annotation (mainly POS and chunking), are developed by Eskin (2000) and Halteren (2000). Volokh and Neumann (2011) employ a method similar to Halteren (2000), to automatically correct errors at the dependency level. Their main idea is to reproduce the gold standard data using MSTParser, MALTParser and MDParser[2]. They use the outputs of any two parsers to detect error nodes, and mark a node as an error if the tag assigned by the two parsers differs from the one in the gold data. In case the assigned tags are different, they use the output of the third parser to check if the tag assigned by the third parser matches with any of the tags assigned by the other two parsers. If the tag matches with any of the other two tags, changes are made in the gold data.

Based on large corpora, van Noord (2004) and De kok et al. (2009) employed error mining techniques. Other examples of detection of annotation errors in Treebanks include Kaljurand (2004) and Kordoni (2003). Most of the aforementioned techniques work well with a large corpora in which the frequency of occurrence of words is very high. None of them work in data sparse conditions except De kok et al. (2009). Moreover, the techniques employed by van Noord (2004), De kok et al. (2009) and Volokh and Neumann (2011) rely on the output of a reliable state-of-the-art parser which may not be available for many languages such as Hindi, the language in our work. Thus, detecting errors in small treebanks is still a challenging task.

Only work by Ambati et al. (2011) and Ambati et al. (2010) tries to detect errors in a Hindi dependency treebank. They used a combination of a rule-based system and a hybrid system to detect the errors. We call this **overall hybrid system**. A rule-based system contributes towards increasing the precision of the system; it uses robust rules formed using annotation guidelines and the framework, whereas

[2] http://mdparser.sb.dfki.de/

a hybrid system is the combination of a statistical module with rule-based post-processing. The statistical module detects potential errors from the treebank and the rule-based post-processing module prunes the false positives[3], with the help of robust and efficient rules to increase the precision of the overall system. A rule-based system is not the same as a 'rule-based post-processing' module as explained in Ambati et al. (2011). Ambati et al. (2011) is different from the Ambati et al. (2010) in that Ambati et al. (2011) uses PBSM (Probability-based Statistical Module) as a statistical module whereas Ambati et al. (2010) uses FBSM (Frequency-based Statistical Module). Ambati et al. (2011) demonstrate that PBSM outperforms FBSM when tested on the same data. Predicting a correct dependency label depends largely on contextual information like information from its parent, sibling, children and grandparent. Current state-of-the-art parsers (MSTParser[4] and MALTParser[5]) use these kinds of features for dependency labeling. Thus, a major limitation of the FBSM is that it can not use more contextual information due to the sparsity issue.

Also there are some limitations of the PBSM. Such as:

- They use smaller feature set and is not effective enough to detect significant amount of the errors in the treebank.
- They use dependency labels as a feature from a current node's context while predicting an appropriate dependency label for the current node. The predicted dependency label might be erroneous because it is being extracted from the same annotated treebank. Then using the same dependency labels as a feature might mislead PBSM into predicting a wrong label for the current node.

## 3 Our Approach

The PBSM proposed by Ambati et al. (2011), extracts some contextual features, trains using gold standard training data that is validated by linguistic experts, creates a model using maximum entropy classification algorithm[6] (MAXENT), tests the system on the testing data and obtains the probabilities for all the possible dependency tags. It then feeds the 1st best and 2nd best tag probabilities into its algorithm to detect errors. For more details of the algorithm please refer to Figure 2 of Ambati et al. (2011). The figure shows the pseudo code of the algorithm. Using this algorithm Ambati et al. (2011) could not only detect

[3] False positives occur when a node that is not an error is detected as an error.
[4] http://sourceforge.net/projects/mstparser/
[5] http://maltparser.org/
[6] http://maxent.sourceforge.net/

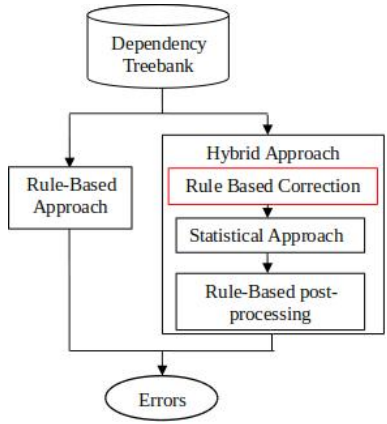


FIGURE 1 Improved version of the overall system proposed by Ambati et al. (2010)

the errors but also classify them into different categories. We use the same algorithm to detect errors with different parameter values.

The kind of features used in a statistical system are very important. Thus, one should carefully choose features which help statistical systems to learn effectively. Ambati et al. (2011) uses parent (POS and CHUNK tag), sibling and child (POS, CHUNK and dependency label) features apart from the node's feature (POS, CHUNK). In this paper, we improve over the PBSM approach proposed by Ambati et al. (2011). We extend the PBSM by adding more linguistically motivated features like dependency labels of the parent, grandparent and count of its children. We call our statistical approach an **Extended PBSM** (**EPBSM**). We compare the performance of the EPBSM with the PBSM. We also improve the hybrid system by placing a new module 'Rule-based correction' before the statistical module as shown in Figure 1. The rule-based correction module is described in detail in Section 3.1 below. We use the EPBSM as the statistical module in the overall hybrid system to compare the performance of the resultant system with the overall hybrid system employed by Ambati et al. (2011). We also show an effect of increasing the training data size on our system. Finally, we perform some user studies which confirm that our system effectively reduces the overall validation time.

### 3.1 Rule-based Correction

The main aim of this module is to detect inter-related errors at the time of validation. Inter-related errors occur when the dependency label of a

node, say 'nodeA', is related to the dependency label of another node, say 'nodeB', in the context of 'nodeA', and the error in the dependency label of 'nodeB' might affect the prediction of the dependency label of 'nodeA' by the statistical module.

The feature set of our statistical module contains dependency labels from the context of 'nodeA' as one of its features and is crucial in detecting errors. So it is better to ensure that the dependency labels are correct while extracting them from the context of 'nodeA'. Hence, we correct them if they follow the rules used for this module. One example of such a case is given below:

**Example1:** In a possible scenario, the dependency label of a node, say 'node1', is 'r6'[7] and the dependency label of its parent node, say 'node2', is something other than 'POF'[8]. We can say that the 'node1' is not an error as it follows the rule that says, that, "if a current node has a dependency label 'r6' and the dependency label of its parent is 'POF' then the dependency label for the current node is wrong, the label should be either 'r6-k1' or 'r6-k2' and vice-versa" [9]. But what if the dependency label of the 'node2' is wrong? This, in turn, would affect the dependency label of 'node1'. If we can somehow detect that the dependency label of 'node2' is 'POF', then according to the rule mentioned above, the dependency label of 'node1' can not be 'r6', it should be either 'r6-k1' or 'r6-k2'. This implies that 'node1' is an error. It is unlikely that the statistical module will detect the error present in 'node1' because of the incorrect contextual information of 'node2'.

To resolve this problem we introduce module in which we place rules like rule the following: "If the dependency label of the current node'node1' is marked as 'k2' and root of its parent node'node2' is 'kara'(do) then the dependency label of the current node must be changed to 'POF'." In this module, while extracting the dependency label of 'node2' as a feature of 'node1', we check if a rule can be applied at 'node2'. If any of the rules are applicable, as in this example, we mark 'node2' as an error. Instead of 'k2', we then use 'POF' as the dependency label of 'node2' as a feature for 'node1'. Since we have now corrected the feature structure of 'node1', the statistical system might detect the error in 'node1'. Note that we do not make changes in the dependency label of the parent in the data, we use a modified label in the feature set of the current node. This module is used before the statistical system to correct the extracted feature set of the current node and to identify additional errors. The additional accounted under the

[7] The genitive/possessive relation which holds between two nouns.
[8] Part of relation such as conjunct verbs.
[9] from the Hindi dependency guidelines (Bharati et al. (2009))

| Approach | Total Error Instances | System Output | Correct Errors | Recall |
|---|---|---|---|---|
| PBSM of Ambati et al. (2011) | 843(7113) | 2000 | 481 | 57.06% |
| EPBSM: Our Approach + rule-based correction | 843(7113) | 2143 | 619 | 73.42% |

TABLE 1 Comparison of performance of statistical model at dependency level

rule-based approach but not under the EPBSM. So error in 'node2' is calculated under the performance of the rule-based approach.

## 4 Experiment and Results

Like Ambati et al. (2011) we use a 65k token manually annotated and validated sample of data (2694 sentences) derived from the Hindi dependency treebank for evaluation. The data is divided into 40k, 10k and 15k for training, development and testing respectively. We use this data to compare our system with Ambati et al. (2011). Later we also train our system with a larger amount of cross-verified[10] data to show its effect on our system's overall performance. We used the training data to train a model and the development data to tune the parameters like threshold values. For our experiments, thres_max= 0.8, thres_min = 0.4, thres_minX = 0.4 and thres_dif = 0.2 gave the best performance.

Table 1 shows the performance of the EPBSM approach with the rule-based correction module and compares it with the performance of the PBSM. The PBSM could detect 57.06% of the dependency error. But using the EPBSM, with an improved feature set, we could detect 63.72 % of the dependency errors. After adding the rule-based correction module in front of the EPBSM the recall of the system increased by 9.68% to 73.4% with a reasonable precision value of 26.45%. Adding more rules to the rule-based correction module can further increase the recall of the EPBSM. There is an overall improvement of 16.34% in the recall value of our approach over the PBSM. Note that the 73.4% of the errors detected by the EPBSM does not contain the additional errors marked by the rule-based correction module. Those errors are accounted for by the performance of rule-based system as in the example1; The additional error in the parent node is accounted under performance of the rule-based system.

Table 2 compares the accuracies obtained by the overall hybrid system employed by Ambati et al. (2011) and their system after replacing

[10] Annotators correcting each other's data but the data is not corrected by experts of the language.

| Approach | Total Error Instances | System Output | Correct Errors | Recall |
|---|---|---|---|---|
| Ambati et al. (2011) overall system with their PBSM | 843(7113) | 2165 | 646 | 76.63% |
| Overall Hybrid system with our EPBSM + Rule-Based correction | 843(7113) | 2187 | 663 | 78.64% |

TABLE 2 Comparison of performance of overall hybrid system at dependency level

| Approach | Total Error Instances | System Output | Correct Errors | Recall |
|---|---|---|---|---|
| Our EPBSM | 843 | 2143 | 619 | 73.42% |
| Our EPBSM (increased training data) | 843 | 2217 | 652 | 77.34% |
| Our overall Hybrid System | 843 | 2187 | 663 | 78.64% |
| Our overall Hybrid System (increased training data) | 843 | 2252 | 687 | 81.49% |

TABLE 3 Performance of our systems after increasing the training data size, our overall hybrid system uses the EPBSM and rule-based correction module

it with our rule-based correction module and the EPBSM approach. Recall of the overall system increased by 2% from the recall obtained by Ambati et al. (2011).

In the overall hybrid system, the rule-based system and the hybrid system detected a total of 170 and 619 error instances respectively. Out of these, 126 error instances are common to both the rule-based system and the PBSM module. This is the reason for small increase of only 2% in the recall of the overall system. But this also implies that learning of the statistical module has improved a great deal due to the addition of new features and the rule-based correction module. Note that our main aim is to achieve a high recall value with a significant precision as false positives can easily be discarded by the validators.

Table 3 compares the performance of our EPBSM and the overall hybrid system obtained before and after increasing the size of the training data. The size of new training data used is around 290k words. We can see that there is a significant rise of 3.92% and 2.85% in the recall of both the EPBSM and the overall hybrid system respectively.

## 5 User Studies

As we discussed earlier manual validation of a treebank takes a lot of time. So it is important to reduce the validation time by suggesting a smaller set of potential error nodes to the annotators. This way they

just need to check the set of suggested potential error nodes for errors.

We performed a user study, in which, annotators were first given 280 sentences containing a total of 2647 nodes from the treebank for manual validation. They were asked to note the time taken to validate each file. Annotators took a total of 30 hours to validate all the sentences i.e.they validated on average 88 nodes in an hour. Then we computed the potential errors using our overall hybrid system on the same set of annotated files given to the annotators. 1128 potential errors nodes were detected by our system containing 81% of the correct error nodes. These potential error nodes were then given again to the same annotators for the manual validation. They were again asked to note the validation time. To validate these potential error nodes, annotators took a total time of 16 hours, i.e. 71 they validated nodes per hour. This is more time taken than the previous 88 nodes per hour because this time the annotators looked at each and every node very carefully as all the nodes were potential error nodes. Although annotators took more time to validate a single node this time, they did not have to go through all the 2647 nodes but only 1128 nodes. We cut down by nearly half the validation time and at the same time corrected 81% of the errors in the treebank.

## 6 A few system improvements to reduce the validation time

Apart from detecting errors automatically, there are a few system improvements that can be used to save additional validation time. We analyzed the whole process which annotators use to detect errors in a treebank. We found that,

- Some of their time is spent in searching for a specific node id in a treebank,
- It also takes time to confirm whether a node is an error as annotators have to look into the whole context of a node.

So we developed a system in which we took care of the issues mentioned above and integrated it with Sanchay. We provide annotators with an error file[11] that the annotators open using Sanchay. When an

[11] It contains the list potential error nodes with different information like the node's Id, contextual information, and the 1st best and 2nd best dependency labels predicted by a statistical module. We also include a comment where we mention if a potential error node is detected using the rule based system or the hybrid system. In case of a rule-based system, we mention the rule behind this; in case of a statistical system, we categorize them into one of the three error categories mentioned in the algorithm.

annotator clicks on the potential error node shown to him/her, the sentence containing that potential error node gets automatically displayed in Sanchay and the node gets highlighted. This way he/she does not have to search for the node manually. We also provide sufficient information (parent's and sibling's dependency labels and POS tags) in the error file which often is enough to decide if the node is an actual error node or not. So for that node, the annotators need not have to go to the sentence at all. These improvements help the annotators and saved a lot of time.

## 7 Conclusion

In this work, we have improved the system proposed by Ambati et al. (2011) and reduced the overall manual validation time. Our EPBSM with the rule-based correction module detected 73.42% of the total errors which is 16.34% more than that of PBSM of Ambati et al. (2011). There is also an increase of 2% in the performance of the overall hybrid system over Ambati et al. (2011) to 78.64%. The increase in the performance of the overall hybrid system is relatively less than the increase in the performance of the EPBSM. This is mainly because there is an overlap between the errors predicted in both the rule-based system and the statistical system. But the important thing is that we have successfully improved the performance of the statistical system using an improved feature set and the rule based correction module. The combined framework can now easily be adopted for any treebank by using similar rules for that language. We have also proposed a system through which these detected errors can easily be corrected in less time.

We can not expect the experts to manually correct a large amount of data as their time is valuable. Thus, we experimented with 290k cross-verified data to increase the size of the training data. We have successfully shown that increasing the training data size also enhances the performance of our system to a high 81.49% although the training data is only cross-verified. So we do not require experts to correct the training data if we can have a large amount of cross-verified data. Apart from detecting a significant number of errors, our approach reduces the overall validation time and streamlines the overall validation process.

## Acknowledgments

We would like to thank Rafiya Begum, Samar Husain, Preeti Pradhan and Nandini Upasani for helping us during the validation process and the user studies. The work reported in this paper is supported by the NSF grant (Award Number: CNS 0751202; CFDA Number: 47.070).

## References

Ambati, B. R., R. Agarwal, M. Gupta, S. Husain, and D. M. Sharma. 2011. Error detection for Treebank Validation. In *The 9th International workshop on Asian Language Resources(ALR)*. Chiang Mai, Thailand.

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

Bharati, A., D. M. Sharma, S. Husain, L. Bai, R. Begam, and R. Sangal. 2009. *AnnCorra: TreeBanks for Indian Languages, Guidelines for Annotating Hindi TreeBank (version â 2.0)*.

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

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

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

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

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

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

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

Kaljurand, K. 2004. *Checking treebank consistency to find annotation errors*.

Kordoni, V. 2003. Strategies for annotation of large corpora of multilingual spontaneous speech data. In *The workshop on Multilingual Corpora: Linguistic Requirements and Technical Perspectives held at Corpus Linguistics(2003)*.

van Noord, Gertjan. 2004. Error mining for wide-coverage grammar engineering. In *ACL(2004)*. Barcelona, Spain.

Volokh, Alexander and Gunter Neumann. 2011. Automatic detection and correction of errors in dependency treebanks. In *ACL-HLT(2011)*. Portland, Oregon, USA.

Xia, F., O. Rambow, R. Bhatt, M. Palmer, and D. M. Sharma. 2009. Towards a multi-representational treebank. In *The 7th International Workshop on Treebanks and Linguistic Theories*. Groningen, Netherlands.