A GUI to Detect and Correct Errors in Hindi Dependency Treebank

Rahul Agarwal[1], Bharat Ram Ambati[1], Anil Kumar Singh[2]

[1]Language Technologies Research Center (LTRC), IIIT-H, India
[2] School of Computer Engg., KIIT University, India
[1]rahul.agarwal,ambati@research.iiit.ac.in, [2]nlprnd@gmail.com

Abstract

A treebank is an important resource for developing many NLP based tools. Errors in the treebank may lead to error in the tools that use it. It is essential to ensure the quality of a treebank before it can be deployed for other purposes. Automatic (or semi-automatic) detection of errors in the treebank can reduce the manual work required to find and remove errors. Usually, the errors found automatically are manually corrected by the annotators. There is not much work reported so far on error correction tools which helps the annotators in correcting errors efficiently. In this paper, we present such an error correction tool that is an extension of the error detection method described earlier (Ambati et al., 2010; Ambati et al., 2011; Agarwal et al., 2012).

Keywords: Treebank, Error Detection, Graphical User Interface

1. Introduction

Availability of large amounts of annotated data is playing a crucial role in developing high accuracy NLP tools like Part-Of-Speech taggers, parsers, machine translation systems. Such data have also proved to be crucial resources for NLP research and for developing solutions for various NLP related applications. To be effectively useful, considering its primary role in providing the appropriate linguistic knowledge to the machine and given the fact that we still do not have good techniques to use 'erroneous' data, in the way that humans can, it is highly desirable that the annotated data should be as free of errors as possible. Hence, the importance of validation of data and error correction cannot be underestimated. On the other hand, annotation itself is a time consuming task and validating the whole data manually would be very time consuming as the validators will have to look at each word in the annotated corpus. To overcome this problem, a number of researchers are working on automatic detection of errors in annotated data.

With validation and correction tools becoming an important part of making treebanks error-free and consistent, efforts are now being been made in this direction to develop error detection tools. One such approach for treebank error detection was employed by Dickinson and Meurers (2003a; 2003b; 2005), where they identified 'variations' in syntactic annotation. They developed a treebank error detection approach based on the idea of 'ngram variation detection'. Using this approach, they tried to find strings which occur multiple times in the corpus, but have varying syntactic annotations. This can happen 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. Adapting from a generalized approach on discontinuous structural annotation, this work was extended to detect errors at the dependency level in treebanks (Boyd et al., 2008).

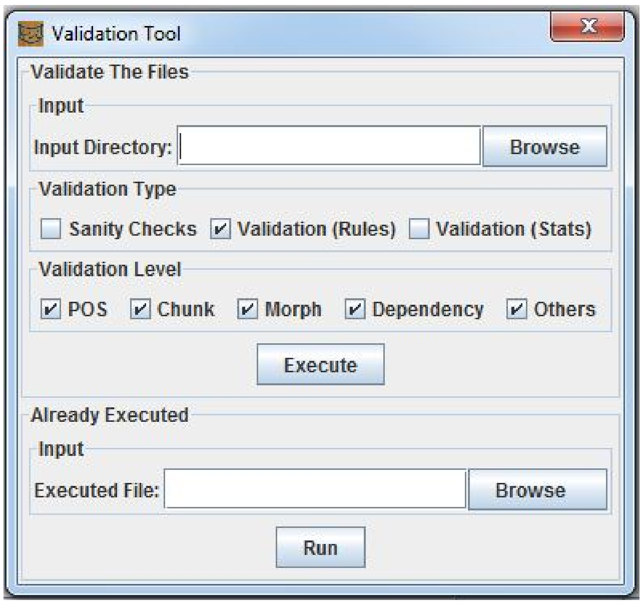
Some other earlier noteworthy methods employed for error detection in syntactic annotation (mainly POS and chunk markup) are by Eskin (2000) and Halteren (2000). Volokh and Neumann (2011) employ a method similar to Halteren (2000) to automatically correct errors at the dependency level. Their main idea was to reproduce the gold standard data using MSTParser, MALTParser and MDParser[1].

There have been a few recent attempts at detecting errors in Hindi dependency treebank by Ambati et al. (2010; 2011) and Agarwal et al. (2012) where they used a hybrid system, a combination of rule-based and statistical approaches to detect errors. The two works are different in terms of the statistical techniques used. Ambati et al. (2010) uses *frequency based statistical approach* where they calculate frequency of various patterns, while Ambati et al. (2011) uses *probability based statistical approach* which tries to predict the best tag for a node given its contextual information as feature to *Maximum Entropy Model.* Agarwal et al. (2012) improves over the Ambati et al. (2011) by using an extra module and an improved feature set. But the main focus of Ambati et al. (2010), Ambati et al. (2011) and Agarwal et al. (2012) is to reduce the validation time of a treebank.

In spite of these efforts in detecting errors in treebanks, there has been a lack of user friendly interfaces that help the validators in detecting errors and correcting them. In this paper, we present such a tool which is being used for developing Hindi dependency treebank (Bhatt et al., 2009; Xia et al., 2009). The Hindi dependency treebank contains information encoded at the morpho-syntactic (morphological, part-of-speech and chunk information) and syntactico-semantic (dependency) levels. Sentence level information like voice type is also annotated for each sentence.

We present a validation tool, using which a human validator can detect errors in the annotated Hindi dependency treebank and correct them. This validation tool uses error detection method employed by Ambati et al. (2010) to detect POS and Chunk related errors while Agarwal et al. (2012) to detect errors dependency related errors in treebank. From different error detection methods provided in the GUI, a

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



Validate The Files

Input

Input Directory:

Validation Type

☐ Sanity Checks ☑ Validation (Rules) ☐ Validation (Stats)

Validation Level

☑ POS ☑ Chunk ☑ Morph ☑ Dependency ☑ Others

Already Executed

Input

Executed File:

Figure 1: Validation Tool Interface

validator can choose any method, run the tool to detect the errors and then correct the error instances manually. Though we use the GUI for detecting errors in Hindi dependency treebank, it can be used for detecting and correcting errors in any treebank, as this tool has been integrated with Sanchay (Singh, 2008; Singh and Ambati, 2010), which includes a user friendly annotation interface for various kinds of morphological, syntactic and semantic annotations.
In section 2, we describe the annotation tool (Sanchay). In section 3, we describe different types of errors in a treebank. Section 4 and 5 explain the functionalities of the GUI in detail. In section 7, we show the performance of the systems used as the backend for the GUI. In section 7, we present the user studies performed involving this GUI. We conclude the paper with section 8.

2. Sanchay

Annotation in our case is being carried out using a tool called Sanchay[2]. Sanchay is an open source platform for working on language data using computers and also for developing Natural Language Processing (NLP) or other text processing applications. Apart from the syntactic annotation interface (used for Hindi dependency annotation), it has several other useful functionalities as well. Encoding conversion, transliteration, corpus annotation API, Sanchay corpus query language (Singh, 2012), parallel corpus annotation, language and encoding detection, n-gram model generation are a few of them. Most of the treebank or syntactic annotation work on Indian languages is being carried out using the Sanchay Syntactic Annotation Interface.

3. Type of Errors in Hindi Treebank

We classify the errors in treebank into two major groups, namely **sanity errors** and **annotation errors**. We shall take an example and see the difference between these two. According to Hindi POS guidelines (Bharati et al., 2006), ‘JJ’ and ‘NN’ are the POS tags for adjectives and common nouns respectively. For an adjective, if an annotator marks the POS tag as ‘ADJ’, a tag which is not present in the predefined list of POS tags in the guidelines for Hindi annotation, we call it a sanity error. But, if the annotator marks that adjective with ‘NN’ tag, a POS tag present in the guidelines but wrong in the context, we call it an annotation error. A list of pre-defined rules is used for detecting sanity errors[3]. Rule-based component of the system comprises of high precision rules which are formed based on the annotation guidelines and the annotation framework[4], whereas the statistical component helps in detecting a wide array of potential errors and suspect cases using counts and probabilities.

4. The Validation Tool

We have developed a Graphical User Interface (GUI) for detecting and correcting the errors as shown in Figure 1. As the Syntactic Annotation Interface of Sanchay is being used for Hindi treebank annotation, we have used the same tool for error correction as well. We have integrated our tool into Sanchay. In this way, an annotator can annotate, detect errors and correct them with Sanchay itself. Figure 1 shows the validation tool interface. A brief description of the functionalities present in the interface is provided in the following sections. Demo of the tool is available at http://web.iiit.ac.in/ rahul_agarwal/ValidationTool/demo.avi.

4.1. Input

There is an input option where we can browse for directory containing the files that need to be validated. Here, we can

[2]http://sanchay.co.in/

[3]Rules being used to detect sanity errors are available at http://web.iiit.ac.in/ rahul_agarwal/ValidationTool/sanityChecks.html.

[4]Rules for rule-based system are available at http://web.iiit.ac.in/ rahul_agarwal/ValidationTool/RuleBasedErrors.html.

give path of a file or a directory of annotated data on which we want to run the tool.

4.2. Validation Type

We can select the type of error detection mechanism using this feature. These types are mentioned below.

4.2.1. Sanity Checks

It runs the sanity checker module to detect the sanity error.

4.2.2. Validation (Rule-based)

It applies the rules from the Hindi annotation guidelines mentioned in section 3.. The rules are written in the Sanchay corpus query language (Singh, 2012). Sanchay runs the queries to detect errors.

4.2.3. Validation (Statistics-based)

It runs the statistical annotation error detection module of Agarwal et al. (2012). The tool also has the flexibility to add statistical model used by Ambati et al. (2010) and Ambati et al. (2011).

4.3. Validation Level

Once we have chosen the validation type, we can also choose a more specific option for validation, i.e., the linguistic level of annotation:

1. **POS:** Part-Of-Speech levels
2. **Chunk:** Chunk level
3. **Morph:** Morphological level
4. **Dependency:** Dependency level
5. **Others:** Other sentence level information like, voice type of the sentence.

For example, if we want to detect sanity errors at POS and chunk levels, then we need to select 'Sanity Checks' from the Validation Type and 'POS' and 'Chunk' from the Validation Level.

4.4. Running the Tool

Finally, press the Execute button. Based on the specified options, the tool runs the corresponding error detection module on the input. Errors detected are presented in a tabular format, which we call as 'error table'.

4.5. Error Table

Figure 2 shows an example of Error Table. There are six columns in the table:

1. **Rule:** Displays the rule (not applicable for 'Validation (Statistics-based)') which is responsible for showing the error along with node and sentence ids.
2. **Matched Node:** The node at which the error occurred.
3. **Context:** The context in which the above node has occurred.
4. **Referred Node:** The parent of the matched node in the dependency tree (This column will be empty if the data is not yet annotated at dependency level).
5. **File:** Path of the file to which the matched node belongs.
6. **Comment:** Displays a possible reason of the error in a running text.

On clicking a cell in the first column of any row, that particular node will be highlighted and displayed using the 'Syntactic annotation interface' of Sanchay. Using this interface, the validator can correct the respective error. We can resize any column in the error table. We can also sort the table based on any preferred column. For example, if we want to correct all the errors in a file before going to next file, we will sort it based on the 'File' column. Similarly, one can correct errors based on the error type, by sorting on 'Comment' column.

4.6. Already Executed

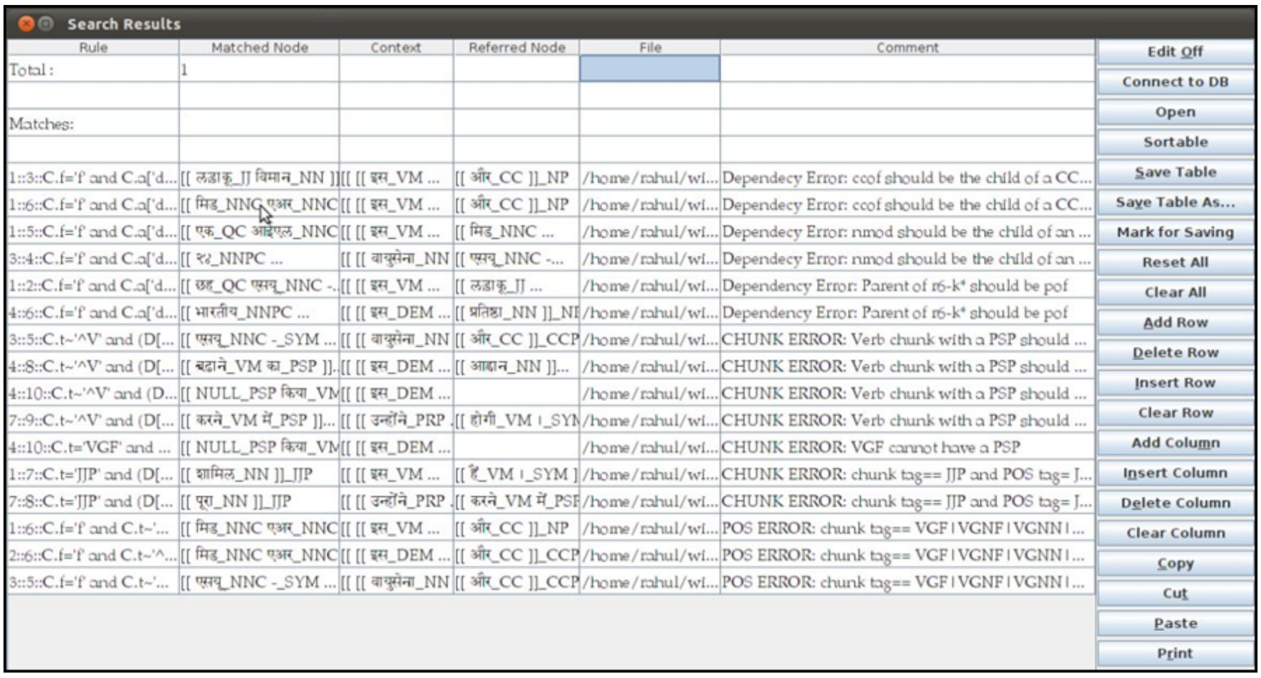
Once the error table is displayed, we can save it, if we wish to make changes afterwards. We can reopen the table by browsing the file from the Already Executed section and pressing the Run button. This functionality is really useful when the error table is very huge. Validator can correct some errors, save rest of the table and correct the remaining errors next time without re-running the tool.

5. Functionalities to Reduce the Validation Time

While developing the GUI for the validation tool, we also concentrated on some small factors that can help in reducing the validation time. From feedback given by the annotators and our interaction with them, we found that some of the annotator's time is spent in searching for a specific node id in a treebank, it also takes time to confirm whether the node is an error as annotators have to look into whole context of the node. We developed the tool in such a way that annotators do not have to do these things manually.
When annotators detect errors using the validation tool, they are displayed with an error table listing all the possible detected errors. When annotator clicks on the potential error node shown to him/her, the sentence containing that potential error node gets automatically displayed in the Sanchay interface and the node gets highlighted. This way he/she does not have to search for the node manually. Also, the Sanchay query result table provide sufficient context information (parent's and sibling's dependency labels and POS tags) in the error file which is usually enough for them to decide if the node is an actual error node or not, so that they may not have to go into the sentence for that node at all in certain cases (e.g. if it is not an error).
All these practices save a lot of extra time and help annotators by reducing the extra work of opening each annotated file and searching for the node manually.

6. Performance of the System Used

For detecting POS and Chunk errors in the treebank, we have used a system employed by Ambati et al. (2010), the performance of which is shown in table 1.
For detecting dependency errors, we have put a system emplpyed by Agarwal et al. (2012). We used this system because it works better than any other error detection system



Rule	Matched Node	Context	Referred Node	File	Comment
Total :	1				
Matches:					
1::3::C.f='f' and C.a['d...	[[लडाकू_JJ विमान_NN]]	[[[[इस_VM ...	[[और_CC]]_NP	/home/rahul/wi...	Dependecy Error: ccof should be the child of a CC...
1::6::C.f='f' and C.a['d...	[[मिड_NNC एअर_NNC	[[[[इस_VM ...	[[और_CC]]_NP	/home/rahul/wi...	Dependecy Error: ccof should be the child of a CC...
1::5::C.f='f' and C.a['d...	[[एक_QC आईएल_NNC	[[[[इस_VM ...	[[मिड_NNC ...	/home/rahul/wi...	Dependecy Error: nmod should be the child of an ...
3::4::C.f='f' and C.a['d...	[[२८_NNPC ...	[[[[वायुसेना_NN	[[एसयू_NNC -...	/home/rahul/wi...	Dependecy Error: nmod should be the child of an ...
1::2::C.f='f' and C.a['d...	[[छह_QC एसयू_NNC -.	[[[[इस_VM ...	[[लडाकू_JJ ...	/home/rahul/wi...	Dependency Error: Parent of r6-k* should be pof
4::6::C.f='f' and C.a['d...	[[भारतीय_NNPC ...	[[[[इस_DEM ...	[[प्रतिष्ठा_NN]]_NI	/home/rahul/wi...	Dependency Error: Parent of r6-k* should be pof
3::5::C.t~'^V' and (D[...	[[एसयू_NNC -_SYM ...	[[[[वायुसेना_NN	[[और_CC]]_CCP	/home/rahul/wi...	CHUNK ERROR: Verb chunk with a PSP should ...
4::8::C.t~'^V' and (D[...	[[बढाने_VM का_PSP]].	[[[[इस_DEM ...	[[आह्वान_NN]]...	/home/rahul/wi...	CHUNK ERROR: Verb chunk with a PSP should ...
4::10::C.t~'^V' and (D...	[[NULL_PSP किया_VM	[[[[इस_DEM ...		/home/rahul/wi...	CHUNK ERROR: Verb chunk with a PSP should ...
7::9::C.t~'^V' and (D[...	[[करने_VM में_PSP]]...	[[[[उन्होंने_PRP .	[[होगी_VM ।_SYI	/home/rahul/wi...	CHUNK ERROR: Verb chunk with a PSP should ...
4::10::C.t='VGF' and ...	[[NULL_PSP किया_VM	[[[[इस_DEM ...		/home/rahul/wi...	CHUNK ERROR: VGF cannot have a PSP
1::7::C.t='JJP' and (D[...	[[शामिल_NN]]_JJP	[[[[इस_VM ...	[[है_VM ।_SYM]	/home/rahul/wi...	CHUNK ERROR: chunk tag== JJP and POS tag= J...
7::8::C.t='JJP' and (D[...	[[पूरा_NN]]_JJP	[[[[उन्होंने_PRP .	[[करने_VM में_PSI	/home/rahul/wi...	CHUNK ERROR: chunk tag== JJP and POS tag= J...
1::6::C.f='f' and C.t~'...	[[मिड_NNC एअर_NNC	[[[[इस_VM ...	[[और_CC]]_NP	/home/rahul/wi...	POS ERROR: chunk tag== VGF | VGNF | VGNN | ...
2::6::C.f='f' and C.t~'^...	[[मिड_NNC एअर_NNC	[[[[इस_DEM ...	[[और_CC]]_CCP	/home/rahul/wi...	POS ERROR: chunk tag== VGF | VGNF | VGNN | ...
3::5::C.f='f' and C.t~'...	[[एसयू_NNC -_SYM ...	[[[[वायुसेना_NN	[[और_CC]]_CCP	/home/rahul/wi...	POS ERROR: chunk tag== VGF | VGNF | VGNN | ...

Figure 2: Error Table: The Sanchay query result table

Validation Level	Recall
POS errors	75%
Chunk errors	62.5%

Table 1: Performance of the POS and Chunk Error Detection System

for Hindi dependency treebank. The performance of this system is shown in table 2.

Approach	Recall
Statistical Model	77.34%
Overall Hybrid Model	81.49%

Table 2: Performance of the Dependency Error Detection System

7. User Feedback

We also performed user studies of the tool that we have presented. The overall study included detection of errors using Hybrid Approach and the manual correction of the errors detected using this GUI. In the user studies, we report that the tool saved around half of the validation time and at the same time, 81% of the errors were also corrected in the treebank. A detailed information about the user studies can be found in Agarwal et al. (2012).

8. Conclusion

We have presented a GUI based tool for a validation of annotated data to detect and correct errors in a treebank. This tool has proved to be very helpful for the annotators during validation of the ongoing Hindi treebank development work. It also helps in reducing a significant amount of validation time.

9. Acknowledgements

We would like to thank Dr. Dipti Misra Sharma and the annotators for their valuable feedback. Part of the work reported in this paper is supported by the NSF grant (Award Number: CNS 0751202; CFDA Number: 47.070).

10. References

R. Agarwal, B. R. Ambati, and D. M. Sharma. 2012. A Hybrid Approach to Error Detection in a Treebank and Its Impact on Manual Validation Time. In *In Proc. of The 10th International workshop on Treebanks and Linguistics Theories (TLT10)*, Heiderberg, Germany.

B. R. Ambati, 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.

B. R. Ambati, 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(ALR09)*, Chiang Mai, Thailand.

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.

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 *The Third Linguistic Annotation Workshop at 47th ACL and 4th IJCNLP*.

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

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

M. Dickinson 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)*.

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

E. Eskin. 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.

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

Anil Kumar Singh and Bharat Ambati. 2010. An integrated digital tool for accessing language resources. In *The Seventh International Conference on Language Resources and Evaluation (LREC)*, Malta, May. The European Language Resources Association (ELRA).

Anil Kumar Singh. 2008. A mechanism to provide language-encoding support and an nlp friendly editor. In *Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP)*, Hyderabad, India. Asian Federation of Natural Language Processing.

Anil Kumar Singh. 2012. A Concise Query Language with Search and Transform Operations for Corpora with Multiple Levels of Annotation. In *The 8th International Conference on Language Resources and Evaluation (LREC)*, Istambul, Turkey.

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

F. Xia, 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.