Caliphate Journal of Science & Technology (CaJoST)

**ISSN:** 2705-313X (**PRINT**); 2705-3121 (**ONLINE**) **Research Article**

**Article Info**

Received: 17th September 2019

Revised: 20th January 2020

Accepted: 21st January 2020

Department of Computer Science, Sokoto State University

***Corresponding author's email:**

Salisumodi@gmail.com

**Cite this:** *CaJoST,* 2020, 1, 37-42

# Boosting Algorithm for Empty Node Recovery in a Sentence

Buhari Wadata, Nura M. Shagari and *Salisu Modi

This paper presents a boosting algorithm for recovering of empty nodes in the analysis of a sentence which are very essential in perceiving syntactic parsing. The inputs of the boosting algorithm are sentences that are converted to parse trees that lacked empty nodes (outputs of broad coverage syntactic parsers such as Charniak's parser and Collin's parser) which are being taken by the algorithm to produce parse trees with different kinds of empty nodes and their antecedents. Evaluation metrics (precision, recall and F-score) were use in order to highlight the differences on performance of recovering empty nodes on the output of the parser with Penn Treebank analysis. The evaluation of the boosting classifier tree (boosting algorithm) on the output of broad coverage syntactic parsers and Penn Treebank achieves high F-score on most types of empty nodes which leads to high parsing accuracy.

**Keywords:** Empty nodes, Boosting algorithm, output of broad coverage syntactic parsers and Penn Treebank.

## 1. Introduction

Empty nodes are nothing but indicators that tell that important information is missing in the syntactic structure of a sentence. Study of empty nodes recovery does not occur only in English language but also in many languages such as Hindi, Arabic, Japanese, and Chinese. They are very relevant in a sentence in knowing the relationship between grammars. Figure 1. shows a parse tree where two empty nodes are indicated, the first empty node * indicates *Aminu failed something* if the sentence is in form of *Aminu failed…,* and second empty nodes indicates *what Aminu failed to register?* Empty nodes also encode additional information about non-local dependencies between words and phrases which is important for the interpretation of construction such as WH-questions and relative clauses. For example, in the noun

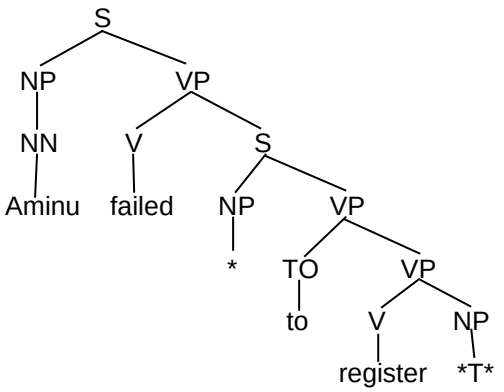


**Figure 1.** Empty nodes indicated in a parse tree

phrase *the game children love* the fact the children are interpreted as the direct object of the verb *love* is indicated in the Penn Treebank by empty nodes and coindexation [2].

Parsing work of Charniak's and Collin's parser have neglected empty nodes in their output based on Penn Treebank [3] that consists much analysis of these empty nodes. Empty nodes play a crucial role for interpretation of semantic structure of a sentence. Previous approaches that tried studying empty node recovery are as follows: In [2] pattern matching algorithm and Penn Treebank are used to recover empty nodes and their antecedents, maximum entropy classifier with mallet toolkit to recover empty nodes in the Chinese Treebank is developed in [5] and Cai *et al.* [1] uses lattice parsing. Furthermore, constituency based parsing algorithm is used [6] to restore empty nodes, while Zhu *et al*. [7] utilized shift reduce constituency parsing. The result for PRO- NP reported by Cai *et al.* [1] is the best among the approaches stated above. One of the main motivations for research on parsing is that syntactic structure provides important information for semantic interpretation; hence syntactic parsing is an important first step in a variety of useful tasks. Broad coverage syntactic parsers mentioned earlier have a good performance, but they typically produce as output a parse tree that only encodes local syntactic information i.e. a tree that does not include any "empty nodes". This work used

  

boosting algorithm and Penn Treebank to put in empty nodes and their antecedents into the output of broad coverage syntactic parsers in such a way that a parsing quality is equally improved.

## 2. Related Work

In [5] maximum entropy classifier with the mallet toolkit to recover empty categories in the Chinese Treebank (CTB) is developed. The classifier scans the words from left to right one by one and determine if there is an empty category before it. When the sentence is paired with its parsed tree, the feature space is all the surrounding words of the target word as well as the syntactic parse for the sentence. The classifier also has access to the empty category labels of all the words before the current word. For the purpose of the presentation they divide their features into lexical and syntactic features. The lexical features are different combination of the words and their parts of speech (POS), while syntactic features are the structural information gathered from the nonterminal phrasal labels and their syntactic relations.

Two different kinds of datasets were used in the evaluation of method in [5]: first, they used the gold standard parse trees from the Chinese Treebank (CTB) as input to their classifier. The version of the parse tree that they used in their classifier is stripped of empty category information. What their system effectively does is to restore the empty categories given a skeletal syntactic parse. Second, they used the Berkeley parser as a representative of the state-of-the-art parsers. The input to the Berkeley parser is words that have already been segmented in CTB. Obviously, for them to achieve fully automatic parsing, they automatically segmented the raw text as well. The Berkeley parser comes with a fully trained model, and to make sure that none of their test and development data is included in the training data in the original model, they retrained the parser with their training set and used the resulting model to parse the documents in the development and test sets. The baseline results using the gold standard trees were 75.3% (precision), 70.5% (recall) and 72.8% (f-score). Using the automatic parses, the results 57.9% (precision), 50.2% (recall) and 53.8% (f-score) respectively. Their results show that given skeletal gold standard parses, empty categories can be recovered with a very high accuracy (close to 90%). They reported promising results (over 63%), when automatic parses produced by an off-the-shelf parser was used as input. One important limitation of this approach is inability to classify empty categories into different types and link them to their antecedents.

Cai *et al.* [1] take a state-of-the-art parsing model, the Berkeley parser [4], train it on data with explicit empty elements and test it on word lattices that can non-deterministically insert empty elements anywhere. The idea is that the state-splitting of the parsing model will enable it to learn where to expect empty elements to be inserted into the test sentences. Unlike the training data, the test data does not mark any empty elements. Cai *et al.* [1] allow the parser to produce empty elements by means of *lattice-parsing,* a generalization of Cocke-Kasami-Younger (CKY) parsing allowing it to parse a word lattice instead of a predetermined list of terminals. Lattice parsing adds a layer of flexibility to existing parsing technology, and allows parsing in situations where the yield of the tree is not known in advance. Lattice parsing originated in the speech processing community and was recently applied to the task of joint clitic-segmentation and syntactic-parsing in Hebrew and Arabic. In their work, they use lattice parsing for empty element recovery.

In [6] transition-based parsing algorithm to incorporate non-local features into the baseline parser is used. In a typical-based parsing process, the input words are put into a queue and partially built structures are organized by a stack. A set of shift-reduce actions are defined, which consume words from the queue and build the output parse. Their method achieves overall f-score of 92.9% in the standard Penn Treebank setup, giving the best results so far for transition based parsing but complexity added to the parse items.

Zhu *et al* [7] proposed shift-reduced constituency parsing by using lexical dependencies. Before conducting lexical dependency extraction, they used the baseline parser to generate constituency parse trees from unlabeled data. Since shift-reduce parsers require POS tags as input, automatic POS tagging was performed on unlabeled data before performing syntactic parsing. For unspaced languages such as Chinese, automatic word segmentation is also needed. They simplified the extraction process by converting automatic parsed constituency trees into dependency trees with Penn2Malt (or other conversion tools). After the tree conversion, the following lexical dependencies are read off from dependency trees, but they restricted the dependencies to those between two words (bigram lexical dependencies) and those between three words (trigram lexical dependencies).

After extracting all lexical dependencies, Zhu *et al* [7] group bigram and trigram lexical

  

dependencies separately into three categories according to their frequencies. Specifically, if a dependency relation is among top-10% most frequent records, then it receives the group tag High Frequency (HF); else if it is in top-20%, then they used the tag Middle Frequency (MF); else they used the tag Low Frequency (LF). Although such a grouping strategy is heuristic in some sense, it has been proven effective. After the grouping, they finally get two lists, containing bigram and trigram lexical dependencies respectively. In their work, $s_i$ denotes the *ith* item from the top of the stack *S*, and $q_i$ the *ith* item from the front end of the queue *Q*. In addition, $s_iw$ ($s_i$ *t*) refers to the head word (POS) of $s_i$ and $q_iw$ ($q_i$ *t*) refers to the word (POS) of $q_i$. Based on the bigram and trigram lexical dependency lists, they proposed a set of dependency features which is described in detail in the following:

### a. Bigram Dependency Features

Bigram dependency features have a generic form of $f_{L/R}$ ($w_1$, $w_2$) which returns a group tag (HF, MF, or LF) if the lexical dependency h$w_1$, $w_2$, $L/R_i$ is found in the bigram lexical dependency list; else it returns *NULL*. The feature template is instantiated into three pairs of features:{ $f_L(s_1w, s_0w)$, $f_R(s_1w, s_0w)$}, { $f_L(s_0w, q_0w)$, $f_R(s_0w, q_0w)$}, and { $f_L(s_1w, q_0w)$, $f_R(s_1w, q_0w)$}. They also combine the features with POS tags of *w*1 and *w*2. Thus they have three more pairs of features in the generic form of $f_{L/R}(w_1, w_2)$ o$t(w_1)$ o$t(w_2)$, where $t(w_i)$ represents the POS tag of the word $w_i$ . All the bigram dependency features are listed in Table 1.

**Table 1.** New features designed on the basis of lexical dependencies. Here the symbol *w* represents a word and the symbol *t* represents a POS tag [7]

| **Bigram Dependency Features** |
|---|
| $f_L(s_1w, s_0w)$ $f_L(s_1w, s_0w)$o$s_1$ to$s_0$ *t* $f_R(s_1w, s_0w)$ $f_R(s_1w, s_0w)$o$s_1$ to$s_0$ *t* |
| $f_L$**(**$s_1w, q_0w$) $f_L(s_1w, q_0w)$o$s_1$ to$q_0$ *t* $f_R(s_1w, q_0w)$ $f_R(s_1w, q_0w)$o$s_1$ to$q_0$ *t* |
| $f_L(s_0w, q_0w)$ $f_L(s_0w, q_0w)$o$s_0$ to$q_0$ *t* $f_R(s_0w, q_0w)$ $f_R(s_0w, q_0w)$o$s_0$ to$q_0$ *t* |
| **Trigram Dependency Features** |
| $_{fL}(s_1w, s_1rdw, s_0w)$ $f_L(s_1w, s_1rdw, s_0w)$o$s_1$ to$s_0$ *t* $f_R(s_1w, s_0ldw, s_0w)$ $f_R(s_1w, s_0ldw, s_0w)$o$s_1$ to$s_0$ *t* |
| $f_L(s_0w, s_0rdw, q_0w)$ $f_L(s_0w, s_0rdw, q_0w)$o$s_0$ to$q_0$ *t* $f_R(s_0w, NONE, q_0w)$ $f_R(s_0w, NONE, q_0w)$o$s_0$ to$q_0$ *t* |

### b. Trigram Dependency Features

Trigram dependency features have the generic form of $f_{L/R}(w_1,w_2,w_3)$. In their method, this feature template is instantiated into two pairs of features. The feature function $f_L$ ($s_1w$, $s_1rdw$, $s_0w$) returns a group tag if h$s_1w$, $s_1rdw$, $s_0w$, $L_i$ is found in the trigram lexical dependency list, where $s_1rdw$ denotes the rightmost modifier of $s_1w$ that has been recognized so far during the shift-reduce parsing process. $s_1rdw$ might be *NONE* if no right modifiers have been recognized for $s_1w$. The other trigram dependency features, $f_R(s_1w$, $s_0ldw$, $s_0w$), $f_L(s_0w$, $s_0rdw$, $q_0w$), and $f_R(s_0w$, *NONE*,$q_0w$) can be explained in a similar way. As with bigram dependency features, POS tags are combined with features to obtain richer feature representations. Trigram dependency features used in their method are summarized in Table1.

Zhu *et al.* [7] experimental results show that the new features achieve absolute improvements over a strong baseline by 0.9% and 1.1% on English and Chinese respectively. Moreover, the improved parser outperforms all previously reported shift-reduce constituency parsers. However, limitations of their approach are: their parser has much smaller time-complexity and relatively low parsing accuracy.

## 3. Methodology

### 3.1 Arcing Game Value (Arc-GV) Boosting Algorithm

Arcing game value (Arc GV) boosting algorithm accepts a training data (training corpus) T = {($x_i$, $y_i$)}$_{i=1}^{L}$ , as its inputs where $x_i$ is an ordered tree and $y_i$ ε {±1} is a class label that each training data is joined with. Arc-GV boosting combines many rules of thumb (weak hypotheses) and can have access to a program for creating weak hypotheses known as the weak learner. For any boosting to be used in solving problem it must be combine with an appropriate weak learner; so this work used decision stump as its weak learner.

**Input**: T = ($x_1$, $y_1$), ($x_2$, $y_2$), . . . , ($x_L$, $y_L$)

where $x_i \in X$, $y_i \in \{-1, 1\}$, Q

**Output:** Parse trees with their margins

1. **Initialization:** $d_i$= 1/L.
2. **for** k = 1 **to Q do**

a. Train base learner using distribution $d_i$

b. Get base classifier $h_{(t_k y_k)} : X \rightarrow \{-1, 1\}$.

c. Calculate the edge $\gamma_k$ :$^{h_{(t_k y_k)}}$ of $\gamma_k = \sum_{i=1}^{L} y_i d_i^k h_{(t_k y_k)}(x_i)$

d. **if** $|\gamma_k| = 1$, **then** $\alpha_r = 0$, for r = 1, . . . , k − 1; $\alpha_k = \text{sign}(\gamma_k)$; break

e. $e_k = \min_{r=1 \ldots k} \gamma_r$

f. Set $\alpha_k = 1/2\log(1+\gamma_k/1-\gamma_k) – 1/2\log(1+e_k/1-e_k)$

g. Updates weights: $d_i^{k+1} = d_i^k \exp(-\alpha_k y_i h_{(t_k y_k)}(x_i))/Z_k$ such that $\sum_{i=1}^{L} d_i^{(k+1)} = 1$

3. $f(x) = \text{sgn}(\sum_{k=1}^{Q} \alpha_k h_{(t_k y_k)}(x))$

4. Return f(x)

: Arc-GV Boosting Algorithm

Arc-GV Boosting iteratively calls decision stumps in a number of rounds so that at round Q, decision stumps would be provided with a set of importance weights over the training corpus. In response, the decision stump takes a parse tree without empty nodes t with its class label y value to compute a hypothesis $h_{(t_k y_k)}$ that maps each instance x value to a real number $h_{(t_k y_k)}(x)$. The sign of this number is interpreted as the predicted class (-1 or +1) of instance x value, while the magnitude $[h_{(t_k y_k)}(x)]$ is interpreted as the level of confidence in the prediction, with larger values corresponding to more confident predictions.

Initially, each training instance in the training corpus will have equal weight and $d_i$ denotes the weight of the ith training instance $(x_i, y_i)$ on the Qth round of boosting., Since hypothesis $h_{(t_k y_k)}$ has been gotten from decision stump. Arc-GV boosting updates the weights of misclassified instance i by multiplying the weight of each instance i value by $\exp(-\alpha_k y_i h_{(t_k y_k)}(x_i))$ and the weight of this instance is increased while the weights of instances that are classified correctly are decreased. In addition, if the confidence of the prediction is higher (that is, the higher the magnitude of $h_{(t_k y_k)}(x_i)$), then the update outcome will be extreme. The weights undergo renormalization, resulting in the rule of the algorithm update. After Q rounds, Arc-GV outputs the final hypotheses f value, which is a linear combination of Q hypotheses produced by the prior weal learners (decision stumps) that is

$$f(x) = \text{sgn}(\sum_{k=1}^{Q} \alpha_k h_{(t_k y_k)}(x))$$

### 3.2 Decision Stumps

Decision stumps are weak classifiers that made their decision by only a single hypothesis or feature. Let t and x be parse tree without empty nodes and tree in the training corpus respectively, and y is a class label (y ε {±1}), a decision stump classifier is given by

$$h_{(t, y)}(x) = \begin{cases} y & t \subseteq x \\ -y & \text{otherwise.} \end{cases}$$

The parameter for classification is the tuple (t, y) , hereafter referred to as the rule of the decision stumps. The decision stumps are trained to find subtree that minimizes the margin for the given training data(training corpus)$T = \{(x_i, y_i)\}_{i=1}^{L}$. When boosting algorithm called the decision stumps in Q rounds, edges will be given by

$$\gamma_k = \sum_{i=1}^{L} y_i d_i^k h_{(t_k y_k)}(x_i) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

and the minimum margin of the tuple $(t_k, y_k)$ is

$$\text{Margin}(t_k, y_k) = \min_{r=1 \ldots k} \gamma_r \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2)$$

Equation (2) indicates that there is Q subtrees which the decision stumps use the margin of the subtrees to select the optimal subtree that is the one with maximum margin among the minimum margins and return it. The margin of the optimal subtree is given by

Margin (optimal subtree)

$$= \max \sum_{k=1}^{Q} \text{margin}(t_k, y_k) \ldots\ldots\ldots (3)$$

### 3.3 Pre-order Traversal for Empty Node Insertion

Decision stumps returned an optimal subtree (tree) which undergoes traversal for empty node insertion and their antecedents. The method used in putting in empty node into a tree (optimal subtree) that does not includes empty nodes is by visiting the subtrees of a tree such that group of patterns at every classified subtree is obtained. Among the group of patterns found if there is an emptyless pattern then pattern with greater ranked would be substituted into the subtree putting an empty node and their antecedents which is the output of the boosting classification's system.

  

## 4. Experiment

The experiment was conducted in a g++ compiler using C++ programming language and the evaluation metrics used for empty node recovery in this system are precision (P), recall (R), and F-score (F) Let H be the number of an empty node identified correctly, J be the total number of an empty node in the Wall Street Journal (WSJ) of Penn Treebank and K be the total number of an empty node reported by the system. Then, the P, R and F are calculated as follows:

$$P = \frac{H}{K}$$

$$R = \frac{H}{J}$$

$$F = 2\frac{PR}{P+R}$$

### 4.1 Training Corpus

Training corpus is a corpus consisting of many sentences in which each sentence has been parsed that is analyzed with syntactic structure. The Wall Street Journal (WSJ) of the Penn Treebank is the training corpus of this work. The trees are extracted from this corpus after the preprocessing of the corpus. This stage of preprocessing ensures that auxiliary verbs such as will, be, have etc and transitive verbs such as leads, walks, sleeps etc are retagged as AX and Vt respectively in all trees in the training corpus, this is because Charniak's parser produced parse trees that do not distinguished auxiliary and transitive verbs tag. Figure 2. shows an instance of how the transitive verb being retagged with Vt.

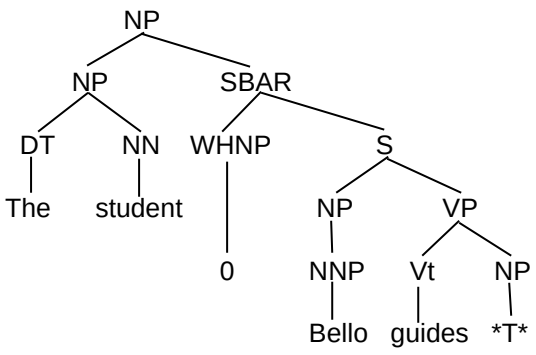


**Figure 2.** Parse tree containing empty nodes

## 5. Results and Discussion

The system is tested using Wall Street Journal (WSJ) of Penn Treebank for 1,000 rounds of boosting with different number of input examples for each empty node and Table 2. shows the summary of the results. The F-score for empty node and its antecedent is very high in both UNIT *U* and SBAR 0: 98.9% and 98.6%. The empty node NP *PRO* has a small precision which leads it to low F-score compared with the remaining ones in the Table 2.

The system achieves an overall of 92.9% precision, 93.2% recall and 93.0% F-score which compared to recent results obtained by the reviewed approaches is greater. Table 2. gives the empty nodes recovery and their antecedents' score for common empty node types using parse trees lacking empty nodes and Wall Street Journal (WSJ) of Penn Treebank.

**Table 2.** The empty nodes recovery and their antecedents' score

| Empty Node Type | Precision (%) | Recall (%) | F-Score (%) |
|---|---|---|---|
| NP *PRO* | 69.10 | 87.20 | 77.10 |
| S *T* | 98.40 | 96.00 | 97.20 |
| NP * | 96.40 | 93.80 | 95.10 |
| NP *T* | 97.30 | 95.30 | 96.30 |
| UNIT *U* | 98.10 | 99.70 | 98.90 |
| WHNP 0 | 97.20 | 89.50 | 93.20 |
| ADVP *T* | 88.70 | 85.70 | 87.20 |
| SBAR 0 | 98.30 | 98.90 | 98.60 |
| **Overall** | 92.90 | 93.20 | 93.00 |

Table 2 shows that the boosting classifier system for empty nodes recovery and their antecedents does quite well, managing an F-score of 93.0% higher than the system in [6] by 0.1%. However, the system also performs better than the Zhu *et al*. [7] system, the difference is quite small: only 2.7%.

NP-NPs and PRO-NPs are empty nodes that are very hard to differentiate due to their resemblance that is why many of the research on parsing find it difficult to recover PRO-NPs competently. The two empty nodes can only be separated by their antecedents that is to say that PRO-NPs have no antecedents while NP-NPs have. The system has very high antecedents recovery on many categories of empty nodes except PRO-NP that does not have antecedent which leads it to low f-score of 77.1% but still is better than Cai *et al*. [1] system by 7.4%.

  

**Table 3.** Comparison of this system with previous approaches

| | This system | Yang and Xue (2010)'s system | Cai *et al*. (2011)'s system | Zhang and Nivre (2011)'s system | Zhu *et al*. (2012)'s system |
|---|---|---|---|---|---|
| **Overall f-score** | 93.00% | 72.80% | 89.30% | 92.90% | 90.30% |
| **NP-PRO f-score** | 77.10% | 66.10% | 69.70% | 67.30% | 67.50% |

Figure 3. shows when the system is compared with Yang and Xue (2010)'s system, there is an increase of F-score from 72.8% to 93.0%. An error of 3.7% is also reduced by the system when compared with Cai *et al*. (2011)'s model.

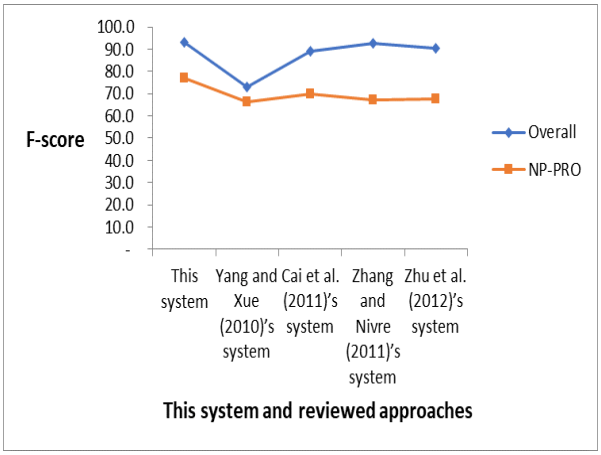


**Figure 3.** Graph of comparison of this system with reviewed approaches

## 6. Conclusion

This work presents a method for enriching the output of broad coverage syntactic parsers (parse trees without empty nodes) with information that is not provided by the parsers themselves, but is available in a Treebank. Using the method with parse trees and Penn Treebank allowed the system to recover empty nodes and their antecedents and also the evaluation metrics were used in order to compare the performance of the systems with the reviewed approaches. The results of the system based on boosting classification model show an overall high F-score of about 93.0%. But the performance for NP-PRO has room for improvement.

## 7. Recommendation

- The investigation of methods for adding information lacked by the output of broad coverage syntactic parsers is needed
- The performance on NP-PRO needs to be enhanced either dependent or independent of the training corpus.

## Conflict of interest

The authors declare no conflict of interest.

## References

[1] S. Cai, D. Chiang, and Y. Goldberg. Language-independent parsing with empty elements. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Stroudsburg, PA, USA. pp. 212-216, 2011.

[2] M. Johnson. A simple pattern matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of 40th Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics,* Stroudsburg, PA, USA, pp. 136-143, 2002.

[3] P. M. Marcus, B. Santorini, and A. M. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Association for Computational Linguistics*, New York, USA. pp. 313–330,1993.

[4] S. Petrov, L. Barrett, R. Thibaux, and D. Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of International Conference on Computational Linguistics*, Stroudsburg, PA. pp. 211-218, 2006.

[5] Y. Yang and N. Xue. (2010). Chasing the ghost: recovering empty categories in the Chinese Treebank. In *Proceedings of international conference on computational linguistics*. Berjing. pp. 1382-1390, 2010.

[6] Y. Zhang and J. Nivre. Transition-based Dependency Parsing with Rich Non-local Features. In *Proceedings of 49th Annual Meeting of the Association for Computational Linguistics.* Portland, Oregon. pp 188-193, 2011.

[7] M. Zhu, J. Zhu, and H. Wang. Exploiting lexical dependencies from large-scale data for better-shift-reduce constituency parsing. In *proceedings of computational linguistics.* Mumbai. pp. 3171-3186, 2012.

  