ONLINE BOOSTING CLASSIFIER TREE FOR EMPTY NODE RECOVERY IN SENTENCE ANALYSIS

Afolayan A.Obiniyi[1], Buhari Wadata[2] and Nura M. Shagari[3]
[1]Ahmadu Bello University Zaria Department of Mathematics
[2,3]Sokoto State University Sokoto Department of Computer Science
aaobiniyi@gmail.com

ABSTRACT
Broad coverage syntactic parsers such as Charniak's parser and Collin's parser produce as output a parse tree that only encodes local syntactic information that is a tree that does not include any empty nodes. This work presents a boosting classifier tree for modification of such parsers to add a wide variety of empty nodes and their antecedents to their parse trees. Evaluation metrics(precision, recall and F-score) were use in order to compare the performance of recovering empty nodes on parser output with the empty nodes annotations in the Penn Treebank. This 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

1. BACKGROUND OF THE STUDY

Empty elements (Empty nodes) in the syntactic analysis of a sentence are markers that show where a word or phrase might otherwise be expected to appear, but does not (Cai et al., 2011). They play an important role in understanding the grammatical relations in the sentence. For example, in the tree of Fig. 1.1, the first empty element (*) marks where *John* would be if *believed* were in the active voice (*someone believed*), and the second empty element (*T*) marks where *the man* would be if *who* were not fronted (*John was believed to admire who?*). Empty elements exist in many languages and serve different purposes. In languages such as Chinese and Korean, where subjects and objects can be dropped to avoid duplication, empty elements are particularly important, as they indicate the position of dropped arguments (Cai et al., 2011).

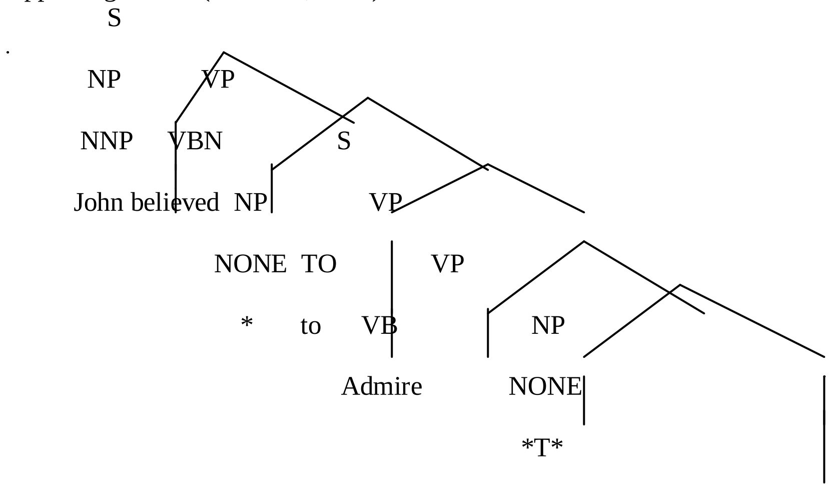

Fig. 1.1 Parse tree with empty elements marked as annotated in the Penn Treebank (Cai et al., 2011).

The Penn Treebank (Marcus et al., 1993) contains detailed annotations of empty elements. Yet most parsing work like Charniak's and Collins's parser based on these resources has ignored empty elements, with some notable exceptions. Johnson (2002) studied empty element recovery in English, followed by several others (Zhang and Clark,2008;Martins et al., 2009; Goldberg and Elhadad, 2010; Zhang and Nivre, 2011; Cai et al., 2011; Zhu et al.. 2012); Recently, empty-element recovery for Chinese has begun to receive attention: Yang and Xue (2010) treat it as classification problem, while Chung and Gildea (2010) pursue several approaches for both Korean and Chinese, and explore applications to machine translation.

The intuition motivating this work is that empty elements are an integral part of syntactic structure, and should be constructed jointly with it, not added in afterwards. Moreover, empty-element recovery is expected to improve as the parsing quality improves. In this work, boosting algorithm will be used to recover the empty elements and their antecedents.

2. MATERIALS AND METHODS

This part consists of sections namely: training corpus, Arcing Game Value (Arc-GV) Boosting Algorithm, Decision Stumps, Pre-order Traversal for Empty Node Insertion and Evaluation Metrics.

2.1 Training Corpus

Training corpus is a text corpus in which each sentence annotated with syntactic structure has been parsed. In this work, a Wall Street Journal (WSJ) of the Penn Treebank is used as a training corpus. The patterns are extracted from this corpus after the preprocessing of the corpus. The preprocessing step relabels auxiliary verbs and transitive verbs in all trees in the training corpus. This relabelling is deterministic and depends only on the terminal (i.e., the word) and its preterminal label. Auxiliary verbs such as *is* and *being* are relabelled as either an AUX or AUXG respectively. The relabelling of auxiliary verbs was performed primarily because Charniak's parser (which produced one of the test corpora) produces trees with such labels. The transitive verb relabelling suffixes of the preterminal labels of transitive verbs with t value. For example, in Fig. 2.1 the verb *likes* is relabeled VBZ_t in this step. A verb is deemed transitive if its stem is followed by an NP without any grammatical function annotation at least 50% of the time in the training corpus; all such verbs are relabelled whether or not any particular instance is followed by an NP. Intuitively, transitivity would seem to be a powerful cue that there is an empty node following a verb.

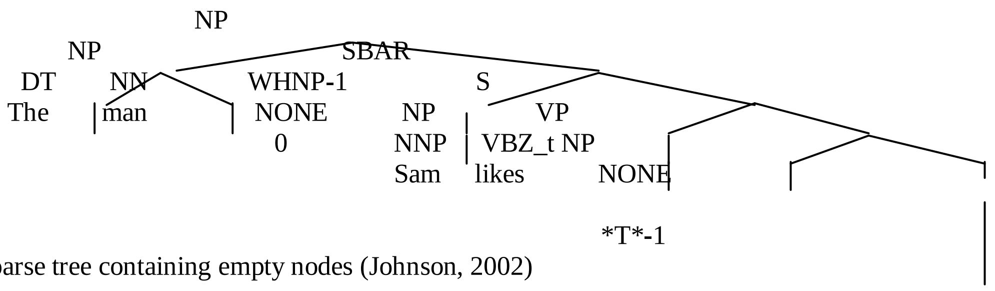

Fig 2.1 parse tree containing empty nodes (Johnson, 2002)

2.2 Arcing Game Value (Arc-GV) Boosting Algorithm

Arcing game value (Arc GV) boosting algorithm takes as its inputs a training data (training corpus) T = $\{(x_i, y_i)\}_{i=1}^{L}$, where $x_i$is a labeled ordered tree and y_i ε {±1} is a class label associated with each training data (the focus here is the problem of binary classification). The idea of Arc-GV boosting is to combine many rules of thumb which are called weak hypotheses. Arc-GV Boosting assumes access to an algorithm or subroutine for generating weak hypotheses called the weak learner. Arc-GV Boosting can be combined with any suitable weak learner; the one used in this work is decision stump.

```
        **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 wth 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 $h_{(t_k, y_k)} : X \to \{-1, 1\}$.
        c. Calculate the edge $\boldsymbol{\gamma}_k$ :$h_{(t_k, y_k)}$ of $\boldsymbol{\gamma}_k = \sum_{i=1}^{L} y_i d_i^k h_{(t_k, y_k)}(x_i)$
        d. **if** $|\boldsymbol{\gamma}_k| = 1$, **then** $\alpha_r = 0$, for $r = 1, . . . , k - 1$; $\alpha_k = \mathrm{sign}(\boldsymbol{\gamma}_k)$; break
        e. $e_k = \min_{r=1 ...k} \boldsymbol{\gamma}_r$
        f. Set $\alpha_k = 1/2\log(1+\boldsymbol{\gamma}_k/1-\boldsymbol{\gamma}_k) - 1/2\log(1+e_k/1-e_k)$
        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. $f(x) = \mathrm{sgn}(\sum_{k=1}^{Q} \alpha_k h_{(t_k, y_k)}(x))$
    4. Return f(x)
```

: Arc-GV Boosting Algorithm

Arc-GV Boosting calls the weak learner (decision stumps) repeatedly in a series of rounds. On round Q, Arc-GV Boosting provides the weak learner with a set of importance weights over the training set. In response, the weak learner 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 example 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 example 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.

The importance weights are maintained formally as a distribution over the training corpus, d_i is used to denote the weight of the ith training example (x_i, y_i) on the Qth round of boosting. Initially, the distribution is uniform. Having obtained a hypothesis $h_{(t_k, y_k)}$ from the weak learner, Arc-GV boosting updates the weights by multiplying the weight of each example I value by exp(-$\alpha_k y_i h_{(t_k, y_k)}$ (x_i)). If$h_{(t_k, y_k)}$ incorrectly classified example i so that $h_{(t_k, y_k)}$ (x_i) and y_i disagree in sign, then this has the effect of increasing the weight on this example, and conversely the weights of correctly classified examples are decreased. Moreover, the greater the confidence of the prediction (that is, the greater the magnitude of $h_{(t_k, y_k)}$ (x_i)), the more drastic will be the effect of the update. The weights are then renormalized, resulting in the update rule of the algorithm. 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 that is

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

2.3 Decision Stumps

Decision stumps are simple classifiers, where the final decision is made 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

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

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

Margin (t_k, y_k) = $\min_{r=1...k} \gamma_r$ (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 $\sum_{k=1}^{Q}$ margin (t_k, y_k) (3)

2.4 Pre-order Traversal for Empty Node Insertion

The optimal subtree returned by the decision stumps is traversed in order to insert empty nodes and their antecedents. The procedure that insert empty nodes into a tree (optimal subtree) not containing empty nodes is as follows: a pre-order traversal of the subtrees of tree is performed and at each subtree the set of patterns that are classified is found. If this set is non-empty the highest ranked pattern in the set is substituted into subtree, inserting an empty node and (if required) co-indexing it with its antecedents which is the output of the boosting classification's system.

The use of a pre-order traversal effectively biases the procedure toward "deeper", more embedded patterns. Since empty nodes are typically located in the most embedded local trees of patterns (that is movement is usually "upward in a tree), if two different patterns corresponding to different non-local dependencies) could potentially insert empty nodes into the same tree fragment in tree, the deeper pattern will be classified at higher node in tree and hence will be substituted. Since the substitution of one pattern typically destroys the context for a classification of another pattern, the shallower patterns are no longer classified.

2.5 Evaluation Metrics

The evaluation metrics used for empty node recovery are: precision (P), recall (R), and f-score (F1). Let H be the number of an empty node identified correctly, J be the total number of an empty node in the training corpus and K be the total number of an empty node reported by the system. Then, the P, R and F1 are calculated as follows:

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

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

$$F1 = 2\frac{PR}{P+R}$$

3 RESULTS AND DISCUSSION

3.1 Results

The system is trained on Wall Street Journal of Penn Treebank for 1,000 rounds of boosting with different number of input examples for each empty node and the results are summarized in Table 3.1. 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 3.1.

The system achieves an overall precision of 92.9%, recall of 93.2% and F-score of 93.0% which is better than the recent results reported by the reviewed approaches. Table 3.1 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 3.1The 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

3.2 Discussion

Table 3.2 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 of Zhang and Nivre (2011) by 0.1%. However, the system also performs better than the Zhu *et al.* (2012)'s system, the difference is quite small: only 2.7%. Two particular cases of interest are NP-NPs and PRO-NPs. The two are only distinguished by their antecedent; NP-NP has an antecedent in the tree, while PRO-NP has none. The system has, for most types of empty nodes, quite high antecedent recovery results, but the difficulty in telling the difference between these two cases results in low F-score for antecedent recovery of PRO-NP, despite the fact that it is among the most common empty node types. Even though this is a problem, this system still does quite well: 77.1% for PRO-NP compared to the 69.7% reported by Cai *et al.* (2011).

Table 3.2 Comparison of this system with previous approaches

	This system	Yang and Xue (2010)'s system	Caiet 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%

Fig. 3.1 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.

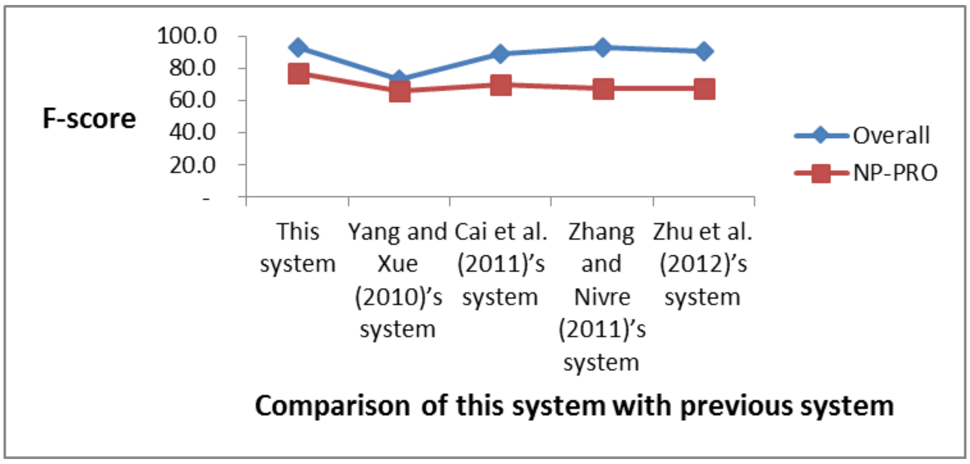

Fig. 3.1 Graph of comparison of this system with previous systems

4 CONCLUSION AND RECOMMENDATION

4.1 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 Wall Street Journal (WSJ) of Penn Treebank allowed the system to recover empty nodes and their antecedents and also the evaluation metrics (precision, recall and F-score) 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.

4.2 Recommendation

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

5. REFERENCES

Cai S., Chiang D., and Goldberg Y. (2011). 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.

Chung T. and Gildea D. (2010). Effects of empty categories on machine translation. In proceedings of the Conference on Empirical Methods in Natural Language Processing, Sapporo, Japan. pp. 80-87

Goldberg Y. and Elhadad M. (2010). An efficient algorithm for easy-first non-directional dependency parsing. In Proceedings of North American Chapter of the Association for Computational Linguistics. Los Angeles, California. pp. 742–750.

Johnson M. (2002). 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.

Marcus P. M., Santorini B. and MarcinkiewiczA. M. (1993). Building a large annotated corpus of English: the Penn Treebank. Association for ComputationalLinguistics, New York, USA. pp. 313–330.

Martins A., Smith N., and Xing E. (2009). Concise integer linear programming formulations for dependency parsing. In Proceedings of Association for Computational Linguistics.Suntec, Singapore. pp 342–350

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

Zhang Y. and Clark S.(2008). A tale of two parsers: investigating and combining graph-basedand transition-based dependency parsing using beamsearch. In Proceedings of theConference on Empirical Methods in Natural Language Processing. Hawaii, USA. pp. 78-87.

Zhang Y. and Nivre J. (2011). 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.

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