International Journal of Engineering & Technology, 7 (4) (2018) 4674-4677

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet. v7i4.14015
Research paper



Transition based parser for telugu language

G. Nagaraju [1] *, N. Mangathayaru [2], B. Padmajarani [3]

[1] Assistant Professor, Department of CSE, VNRVJIET, Hyderabad, TS, India
[2] Professor, Department of IT, VNRVJIET, Hyderabad, TS, India
[3] Professor, Department of CSE, JNTU-CEH, Hyderabad, TS, India
**Corresponding author E-mail: nagaraju_g@vnrvjiet.in*

Abstract

This paper explains transition dependency parsing approaches to build a dependency parser for Telugu language. Telugu treebank is given as an input to transition dependency parsers. One of the best transition dependency parser is the Malt parser. It is an independent system and it has nine methods to parse a sentence of any language. We have applied the treebank on all the methods of a Malt parser among which Arc-eager parser produces state-of-art results for Telugu language. Arc-eager method was produced LA (Label Accuracy) of 63%, UAS (Unlabeled Attachment Score) of 88.1% and LAS (Label Attachment Score) of 62.3%. In this paper we discuss a brief introduction of all Malt Parsing methods and an in detail explanation of Arc-eager dependency parsing.

Keywords: *Dependency Parsing; Telugu Language; Transition Based Parsing; Malt Parser; Arc-Eager Parsing.*

1. Introduction

Telugu language is one of the Dravidian languages. It is a free word order language because if we interchange the words in a sentence still it preserves its meaning and it is morphologically rich. Parser has widely many applications like machine translation and text summarization. Parsing a particular language requires a lot of resources in that language. A lot of resources are available for English and some other languages, but a few resources are available for Indian languages. We shouldn't use the parsing approaches to Indian languages, which are applied to parse a sentence in English language [6] because these languages have their own language structures. English is neither morphologically rich nor free word order language, whereas Indian languages are morphologically rich and free word order. English has a phrase structure [7] whereas Telugu has a dependency structure. The languages which are morphologically rich and free word order structure are parsed by the dependency parsers. So we should apply dependency parsing techniques to parse a Telugu Sentence.

A simple parser was developed for Indian languages based on karaka approaches [8]. Karaka approach was invented by Panini. Panini is an ancient Sanskrit linguist and grammarian. According to his philosophy, each word in a sentence should have a role, is called a karaka. There are 6 karaka's, they are Karta, Karma, Kriya, Apaadana, Sampradaana and Karana so each word of a sentence is mapped to one of these karaka's[6]. This approach requires more linguistic knowledge to map words of a sentence into its corresponding karaka's.

Dependency framework is the best approach for parsing a Telugu language. Dependency Parser represents the relationship between words in a sentence. It uses annotated corpora called treebank, which consists of words of a sentence, POS (parts of speech) tag and its dependent word in a sentence, this structure helps to find the relationship between the words of a sentence, it is used to train a model, after training it builds a model for Telugu Language. Dependency parsers are divided into three categories. They are local or greedy based approaches, globally optimization techniques and hybrid techniques. Malt parser follows local or greedy approach [18], MST parser follows global optimization methods [19] and Sagae and Lavie proposed hybrid techniques [20].

Dependency parsers are either transition based dependency parsers or graph based dependency parsers [2]. Most popular transition based dependency parser is the Malt parser and the best graph based dependency parser is the MST (Maximum Spanning Tree) parser.

Malt and MST parsers use treebank to train a model for a given language and these are language independent systems [2]. In each language, sentences are categorized into two types, they are projective and a non-projective. In a projective sentence, dependency edges aren't cross over each other where as in a non-projective sentence, dependency edges may cross over each other. Malt and MST parsing methods are used to parse both the projective and non-projective sentences [3].

Initially MST parser uses Eisner's algorithm [13] to parse non projective sentences, which has a time complexity of $O(n^3)$, and the method modified by Chu-Liu[14]-Edmonds [15] to non-projective sentences which yields a time complexity of $O(n^2)$. McDonald and Pereira [14] were developed non-projective dependency tree using MST using Online Large Margin learning. This method was produced better results as compare to the former one.

In this paper we discuss introduction to dependency parser and Malt Parser in section 2. Section 3 describes the brief explanation of Arc-eager dependency parsing. Feature selection is discussed in section 4. Results comparison is explained in section 5 and section 6 explains the conclusion and future work.

2. Dependency parsing

Popular transition dependency Parsers are Malt Parser, MST Parser, Turbo Parser, Easy first Parser and ZPar Parser. Malt parser is a transition dependency parser and it uses stack and input buffer to parse a sentence. MST parser is a graph based dependency parser

Copyright © 2018 G. Nagaraju et. al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

and it was developed on the concept of building a maximum spanning tree from a directed graph. It uses Chu-Liu-Edmonds and Eisner algorithms while constructing a maximum spanning tree [17]. Turbo parser uses integer linear programming [9], Easy first parser uses shift reduce method [10], it made arcs randomly either from left or right [21] and ZPar parser uses Malt parser with beam search [11]. Among these parsers Malt parser has produced state-of-art results for Telugu language [12].

2.1. Transition based dependency parsing – malt parser

Malt parser is a transition based dependency parser. It has a stack (s), buffer (b) and arc-set (A). it is used to predict a transition sequence from a given configuration [1]. A configuration is the present status of stack, buffer and arc-set. Stack contains processed words of a sentence, buffer contains unprocessed words and arc-set consists of left-arcs and right-arcs between the words of a sentence. Malt Parser has nine methods, they are Nivre-eager(Arc-eager), Nivre-standard, Covington projective, Covington non-projective, stack projective, stack eager, stack lazy, planar and 2-planar. Covington projective and stack projective methods parse projective sentences and the remaining methods parse both projective and non-projective sentences.

Telugu treebank was applied to all the methods of a Malt Parser. Among all the methods Nivre eager produces best results for a Telugu language. Arc eager is a popular method in dependency parsing to parse a sentence. Configurations are the status of a stack, buffer and arc-set during training a Telugu treebank so each configuration (C) is represented with a set (s, b, A). In the initial configuration (C_s) stack is empty, buffer contains all the words of a sentence and an empty arc set. Initial configuration (C_s) is shown below.

Initial configuration C_s = (S = w_1 , w_2,…….w_n) = ([ROOT], [w_1 , w_2,…….w_n], Φ)

In the terminal configuration (C_t), top of stack contains root word of a sentence, buffer is an empty set and an arc-set contains all the gold-arcs [4]. Terminal configuration (C_t) is shown below.

Terminal configuration (C_t) = ([ROOT], Φ, A)

3. Nivre ARC eager dependency parsing

Arc-eager method contains a stack, buffer and arc-set. Stack contains the processed words, buffer consists of unprocessed words of a sentence, and arc set consists of left-arc and right-arc. This methods has four transitions, they are one Left-Arc 2. Right-Arc 3. Shift 4. Reduce [4]. These descriptions transitions are described below.

1) Left-Arc (label) – There is an arc from first element in the buffer to top of the stack. Pop s_i from top of the stack. Add an arc s_i b_j with label as (s_i, label, b_j) to A. Left arc is possible only If there are no dependencies to s_i from buffer and s_i will not be head for the words in the buffer.
2) Right-Arc (label): It represents there is an arc from top of the stack to the first word in the buffer. Add an arc si → bj with label as (si, label, bj) to A. Right arc is possible only there were no dependencies to bj from buffer and stack, there are no dependencies to si from stack.
3) Shift: Move first element of buffer to the top of the stack.
4) Reduce Pop-top of the stack.

Arc eager parsing uses a static oracle to parse sentence, the following is the Arc eager parsing method.

Algorithm Arc-eager-parsing(s, b, A) returns an action t

If c = (s|i, j|b, A) and (j, l, i) ∈Agold then
  T ← LEFT-ARC (l)
Else if c = (s|i, j|b, A) and (i, l, j) ∈Agold then
  T ← RIGHT-ARC (l)
Else if c = (s|i, j|b, A) and ∃k [k < i ∧∃l [(k, l, j) ∈Agold
∨ (j, l, k) ∈Agold]] then
  T ← REDUCE
Else
  T ← SHIFT
Return t

Fig. 1: Arc-Eager Parser Algorithm to Parse a Telugu Sentence.

The above algorithm is called a static oracle, where c represents a configuration, t represents a transition and Agold represents a arcs in gold tree bank. Static oracle has two drawbacks, First, it cannot handle spurious ambiguities that is, in some situations this algorithm produces two transitions, shift and reduce. In such cases this algorithm always selects a reduce action, but the correct action is shift. Second, sometimes it produces a canonical transition sequence from which we cannot built a correct dependency tree. Because of the above limitations the classifier fails to produce the correct canonical sequence. To avoid these problems we consider non-gold configurations during training. It generates error propagation, that algorithm called as dynamic oracle.

The differences between static oracle and dynamic oracle are that, static oracle always restricts unique transition sequence where as a dynamic oracle did not restrict to a unique transition, where multiple transition sequences are possible. It allows all possible transition sequences, from which it selects the best transition sequence. Dynamic oracle uses zero cost transitions to select the best transition sequence.

An oracle allows multiple transition sequences for a given tree and makes optimal predictions in all configurations. It defines a relation from configurations to transitions. It is represented with o(t;C;G_{gold}) where t is a transition, C is a configuration and G_{gold} is a gold tree. It is a Boolean function which returns one if t is optimal with respect to $G_{gold,}$ otherwise it returns zero. Argmax is used to select the best transition.

tp ← Argmax t∈ZERO_COST w.$\emptyset(c,t)$

Here c is the configuration, t is the transitions, t_p is the predicted transition and ZERO_COST is the possible transitions, obtained from dynamic oracle. Argmax is the maximum function it selects only one transition from multiple transitions which are derived from dynamic oracle. If the predicted transition t_P is different from the transitions in ZERO_COST then weights should be updated. It represents an error, that error is back propagated. Otherwise no need to update the weights. These weights are used to build the model.

Figure 2 represents dynamic oracle uses perceptron algorithm. For a given sentence, it should give a configuration and check whether it is a terminal configuration, if not it finds the predicted transition t_p from a configuration c and also it finds transitions t_o using dynamic oracle. If predicted transition t_p is in t_o then there is no update of model weights. If t_p is not in the t_o then update model weights. This procedure is applied to all the configurations of a sentence until a sentence configuration is reached to a terminal configuration.

4. Feature selection

The features for malt parser are taken from stack, input buffer and POS tags. We have taken 29 features to train a model these features are shown in the following figure 3.

Malt parser uses structured perceptron to train a sentence and LIBSVM and LIBLINEAR methods are used to learn a model. We used LIBLINEAR method to learn a model.

$w \leftarrow 0$
for $i = 1 \rightarrow$ ITERATIONS **do**
  for sentence x with gold tree G_{gold} in corpus **do**
    $C \leftarrow C_s(x)$
    while C is not terminal **do**
      $t_p \leftarrow \text{argmax}_t \, w \cdot \phi(C,t)$
      ZERO_COST $\leftarrow \{t | o(t;C,G_{gold}) = \text{true}\}$
      $t_o \leftarrow \text{argmax}_{t \in \text{ZERO_COST}} \, w \cdot \phi(C,t)$
      if $t_p \notin$ ZERO_COST **then**
        $w \leftarrow w + \phi(C,t_o) - \phi(C,t_p)$
      $t_n \leftarrow$ CHOOSE_NEXT$(i,t_p,$ZERO_COST$)$
      $C \leftarrow t_n(C)$
return w

function CHOOSE_NEXT$_{AMB}(i,t,$ZERO_COST$)$
  if $t \in$ ZERO_COST **then**
    return t
  else
    return RANDOM_ELEMENT(ZERO_COST)

function CHOOSE_NEXT$_{EXP}(i,t,$ZERO_COST$)$
  if $i > k$ and RAND() $> p^4$ **then**
    return t
  else
    return CHOOSE_NEXT$_{AMB}(i,t,$ZERO_COST$)$

Fig. 2: Dynamic Oracle for Arc Eager Dependency Parsing.

Unigram features
s0w, s0p,s1w, s1p,s10p, s0wp, s1wp, B0w, b0p, b10p, b1w, b1p, b2w, b2p, b0wp, b1wp, b2wp
Bi-Gram Features
S0wp_b0wp, s0wp_b0w, s0w_b0wp, s0wp_b0p, s0p_b0wp, s0w_b0w, b0p_b1p
Tri-Gram Features
B0p_b1p_b2p, s0p_b0p_b1p, sh0p_s0p_b0p, s0p_s10p_b0p, s0p_sr0p_b0p, s0p_b0p_b10p

Fig. 3: Feature Selection for Arc-Eager Parsing Method.

4.1. Tree bank

A treebank is an annotated corpus for a language, in which each line contains a word of a sentence. Each word has 10 fields, they are line number, in this sentence starting word always at line number 1, word form, lemma, universal POS (Part of Speech) tag, XPOS (language specific POS), morphological features, Head of the word, Dependency relation with the head, enhanced dependency graph and any miscellaneous annotation.
Telugu Sentence: ceVttu peVrigi pEki powuMxi
Treebank Notation:

1) ceVttu ceVttu NP NN vib-0|tam-04 k1
2) peVrigi peVrugu VGNF VM vib-i|tam-I 4 vmo
3) pEki pEki NP NST vib-0|tam-0 4 vmod
4) powuMxi po VGF VM vib-wA|tam-wA 0 main

4.2. Experiments

Telugu Sentence: nAku kalalo xevawa kanipiMciMxi
In arc eager dependency parsing <root> node is appended at the beginning of the sentence. Now the augmented sentence is
<root> nAku kalalo xevawa kanipiMciMxi
The following figure 4 represents Arc eager parsing steps to parse a Telugu sentence and figure 5 represents a dependency parser notation for a given Telugu sentence.

5. Results

The following table shows the results of all parsing methods. Results are measured in LA, UAS and LAS. LA represents Labelled Accuracy, is the percentage of the labels correctly assigned. UAS represents Unlabelled Attachment Score, is the percentage of heads correctly assigned. LAS represents Labelled attachment score, is the percentage of correct labels to correct heads. We have taken 1300 Telugu sentences, which are annotated. Arc-eager parsing has given best results among all the methods. Nivre eager also known as Arc-eager has produced best results in Labelled Accuracy, Covington projective has produced best results in Unlabelled Attachment score and Again Arc-eager has produced best results in Labelled Attachment Score. Covington Projective has produced least results in Labelled Accuracy, Planar has produced least results in Unlabelled Attachment Score and Covington projective produced least results in Labelled Attachment Score. Overall Arc-eager performance is better even though it has less performance in Unlabelled Attachment Score. There is a variation in UAS and LAS scores because most of the heads were correctly identified as compared to the labels.

Transition	Stack	Buffer	Arcs
	[ROOT]	[nAku kalalo ……]	Φ
SHIFT	[ROOT nAku]	[kalalo xevawa….]	Φ
SHIFT	[ROOT nAku kalalo]	[xevawa kanipiMciMxi]	Φ
SHIFT	[nAku kalalo xevawa]	[kanipiMciMxi]	Φ
LEFT-ARC(K2)	[ROOT nAku kalalo]	[kanipiMciMxi]	A U K2(kanipiMciMxi , xevawa)
LEFT-ARC(K7t)	[ROOT nAku]	[kanipiMciMxi]	A U k7t(kanipiMciMxi , kalalo)
LEFT-ARC(K1)	[ROOT]	[kanipiMciMxi]	A U K1(kanipiMciMxi , nAku)
RIGHT-ARC(main)	[ROOT kanipiMciMxi]	[ ]	A U main(ROOT, kanipiMciMxi)
REDUCE	[ROOT]	[ ]	

Fig. 4: Arc-Eager Parsing Steps to Parse A Telugu Sentence.

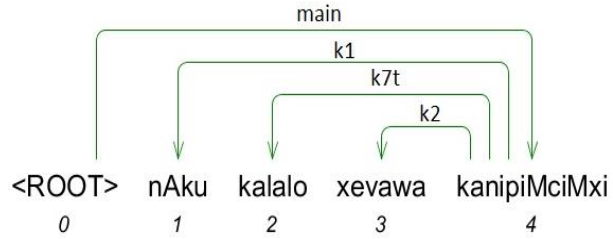


Fig. 5: Dependency Parsing.

Table 1: Comparison of Parsing Methods for Telugu Treebank

S.No	Parsing Algorithm	LA	UAS	LAS
1	Nivreeager	63.1	88.1	62.3
2	Nivrestandard	61.4	87.8	60.9
3	Covproj	61.3	88.3	60.1
4	Covnonproj	62.8	88.5	61.6
5	Stackproj	61.4	87.5	60.9
6	Stackeager	61.8	88.0	60.8
7	Stacklazy	61.9	88.0	61.1
8	Planar	61.9	86.0	60.9
9	2planar	61.6	87.1	60.6

For the measurements LA and LAS Nivre Arc-eager method has given best results, the results have showed in figure 6 and in Figure 7.

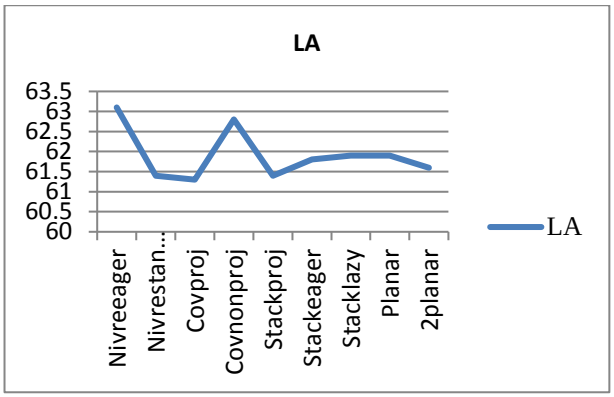


Fig. 6: Label Accuracy of Parsing Methods.

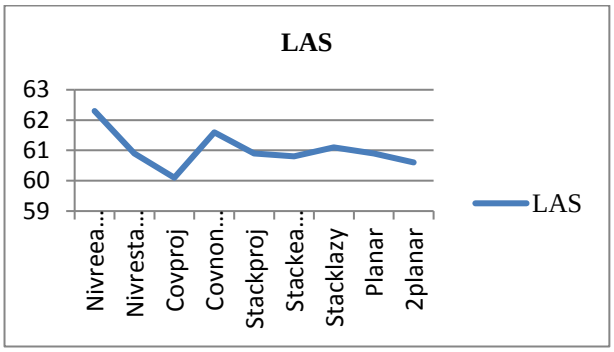


Fig. 7: Label Attachment Score of Parsing Methods.

For the measurement UAS Covington Non-Projective has given best results and the results have showed in figure 8.

6. Conclusion

This paper has explained transition dependency parser to parse a Telugu language also discussed Arc-eager dependency parsing. Construction of Telugu treebank and its fields were discussed. Explored all the dependency parsing methods and discussed their results. Among all the methods Arc-eager Parsing has produced state-of-the-art results for Telugu language.

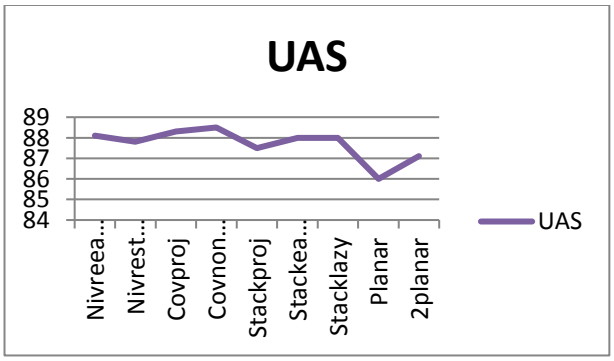


Fig. 8: Unlabeled Attachment Score of Parsing Methods.

7. Future directions

Arc-eager parsing method is the best parsing technique for morphologically rich and free word order languages. Telugu language is such type of language, so Arc-eager parsing method has produced better results and the results should be increased, if the parser uses language specific features like case markers and grammatical agreement along with the standard features. We have used small amount of data set and also the results will be increased by increasing the data set.

References

[1] Nivre, J. (2009) Parsing Indian Languages with MaltParser. In Proceedings of the ICON09 NLP Tools Contest: Indian Language Dependency Parsing, 12-18.

[2] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryiğit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95-135, 2007.

[3] Joakim Nivre, Ryan McDonald (2011) Analyzing and integrating dependency parsers. Computational Linguistics, 37(1):197-230, 2011. https://doi.org/10.1162/coli_a_00039.

[4] Yoav Goldberg and Joakim Nivre. A dynamic oracle for arc-eager dependency parsing. In Proceedings of the 24th International Conference on Computational Linguistics (COLING), pages 959-976, 2012.

[5] Covington, M. A. 2001. A fundamental algorithm for dependency parsing. In Proceedings of the 39th Annual ACM Southeast Conference, pp. 95–102.

[6] G. Nagaraju, N. Mangathayaru, B. Padmaja Rani. Dependency Parser for Telugu language. In Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies, Article No. 138, 2016. https://doi.org/10.1145/2905055.2905354.

[7] Ryo Nagata, Keisuke Sakaguchi. Phrase Structured Annotation and Parsing for Learner English. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, 1837-1847, 2016.

[8] Akshar Bharati, Mridul Gupta, Vineet Yadav, Karthik Gali and Dipti Misra Sharma. Simple Parser for Indian Languages in a Dependency Framework. In Proceedings of the Third Linguistic Annotation Workshop, ACL-IJCNLP, 162-165, 2009.

[9] Martins, N. Smith, E. Xing, Concise Integer Linear Programming for Dependency Parsing. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the fourth International Joint Conference on Natural Language Processing of the AFNLP, pp. 342-350.

[10] Goldberg, Y., Elhadad, M., 2010a. Easy first dependency parsing of Modern Hebrew. In: Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically Rich Languages. SPMRL '10. Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 103–107.

[11] Y. Zhang, S. Clark, Syntactic processing using the generalized perceptron and beam search Computational Linguistics, 37 (1) (Mar. 2011), pp. 105-151. https://doi.org/10.1162/coli_a_00037.

[12] B. Venkata Seshu Kumari, Ramisetty Rajeshwara Rao, Telugu dependency parsing using different statistical parsers. Journal of King Saud University-Computer and Information Sciences.2017, pp. 134-140.

[13] J. Eisner. 1996. Three new probabilistic models for depedency parsing. An exploration. In Proc. COLING.

[14] Y.J. Chu and T.H. Liu. 1965. on the shortest arborescence of a directed graph. Science Sinica, 14:1396-1400.

[15] J. Edmonds. 1967. Optimum branches. Journal of Research of the National Bureau of Standards, 71B:233-240. https://doi.org/10.6028/jres.071B.032.

[16] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In Proceedings of HLT/EMNLP-2005. Vancouver, Canada, pages 523-530.

[17] Ryan McDonald and Fernando Pereira 2006. Online Learning of Approximate Dependency Parsing Algorithms. In Proceedings of 11th International Conference of the European Chapter of the Association for Computational Linguistics.

[18] Joakim Nivre, Johan Hall, and Jens Nillson. 2006. Malt Parser: A data-driven parser-generator for dependency parsing. In proc. of LREC.

[19] Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In Proc of ACL. https://doi.org/10.3115/1219840.1219852.

[20] Kenji Sagae and Alon Lavie. 2006b. Parser combination by reparsing. In Proc of NAACL. https://doi.org/10.3115/1614049.1614082.

[21] Yoav Goldburg and Michael Elhabad. An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. The 2010 Annual Conference of the North American Chapter of the ACL, Pages 742-750, Los Angles, California, June 2010.