Review of Syntactic Parsers and Dependency Parsing

Anuj Yadav[1] Aman Sharma[2]
[1,2]B.Tech Student(7TH Sem.)
[1,2]Department of Electronics and Computers Engineering
[1,2]Dronacharya College of Engineering, Gurgaon, India.

Abstract— *we present a review of novel parser combination scheme that works by reparsing input sentences once they have already been parsed by several different parsers. We apply this idea to dependency and constituent parsing, generating results that surpass state-of-theart accuracy levels for individual parsers. Dependency parsing has been a prime focus of NLP research of late due to its ability to help parse languages with a free word order. Dependency parsing has been shown to improve NLP systems in certain languages and in many cases is considered the state of the art in the field. The use of dependency parsing has mostly been limited to free word order languages, however the usefulness of dependency structures may yield improvements in many of the word's 6,000+ languages. I will give an overview of the field of dependency parsing while giving my aims for future research. Many NLP applications rely heavily on the quality of dependency parsing. For this reason, I will examine how different parsers and annotation schemes influence the overall NLP pipeline in regards to machine translation as well as the baseline parsing accuracy.*

Keywords— NLP, PCFG-based frameworks, McDonald, Pereira

I. INTRODUCTION

Parsing technologies have improved considerably in the past few years, and high-performance syntactic parsers are no longer limited to PCFG-based frameworks (Charniak, 2000; Klein and Manning, 2003; Charniak and Johnson, 2005; Petrov and Klein, 2007), but also include dependency parsers (McDonald and Pereira, 2006; Nivre and Nilsson, 2005; Sagae and Tsujii, 2007) and deep parsers (Kaplanet al., 2004; Clark and Curran, 2004; Miyao and Tsujii, 2008). However, efforts to perform extensive comparisons of syntactic parsers based on different frameworks have been limited. The most popular method for parser comparison involves the direct measurement of the parser output accuracy in terms of metrics such as bracketing precision and recall, or dependency accuracy. This assumes the existence of a gold-standard test corpus, such as the Penn Treebank (Marcus et al., 1994). It is difficult to apply this method to compare parsers based on different frameworks, because parse representations are often framework-specific and differ from parser to parser (Ringger et al., 2004). The lack of such comparisons is a serious obstacle for NLP researchers in choosing an appropriate parser for their purposes.

In this paper, we present a comparative evaluation of syntactic parsers and their output representations based on different frameworks: dependency parsing, phrase structure parsing, and deep parsing. Our approach to parser evaluation is to measure accuracy improvement in the task of identifying protein-protein interaction (PPI) information in biomedical papers, by incorporating the output of different parsers as statistical features in a machine learning classifier (Yakushiji et al., 2005; Katrenko and Adriaans, 2006; Erkan et al., 2007; Sætre et al., 2007). PPI identification is a reasonable task for parser evaluation, because it is a typical information extraction (IE) application, and because recent studies have shown the effectiveness of syntactic parsing in this task. Since our evaluation method is applicable to any parser output, and is grounded in a real application, it allows for a fair comparison of syntactic parsers based on different frameworks.

Parser evaluation in PPI extraction also illuminates domain portability. Most state-of-the-art parsers for English were trained with the Wall Street Journal (WSJ) portion of the Penn Treebank, and high accuracy has been reported for WSJ text; however, these parsers rely on lexical information to attain high accuracy, and it has been criticized that these parsers may overfit to WSJ text (Gildea, 2001; 46Klein and Manning, 2003). Another issue for discussion is the portability of training methods. When training data in the target domain is available, as is the case with the GENIA Treebank (Kim et al., 2003) for biomedical papers, a parser can be retrained to adapt to the target domain, and larger accuracy improvements are expected, if the training method is sufficiently general. We will examine these two aspects of domain portability by comparing the original parsers with the retrained parsers.

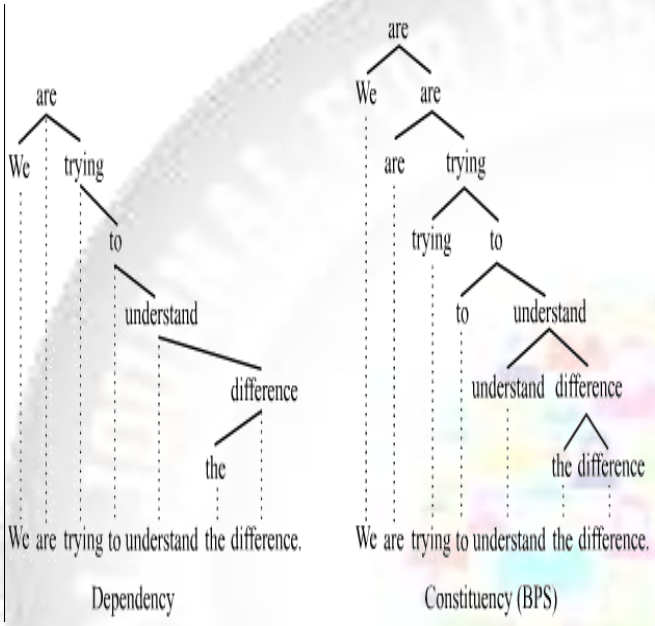
II. SYNTACTIC PARSERS AND THEIR REPRESENTATIONS

This paper focuses on eight representative parsers that are classified into three parsing frameworks: dependency parsing, phrase structure parsing, and deep parsing. In general, our evaluation methodology can be applied to English parsers based on any framework; however, in this paper, we chose parsers that were originally developed and trained with the Penn Treebank or its variants, since such parsers can be re trained with GENIA, thus allowing for us to investigate the effect of domain adaptation.

A. Dependency parsing

Dependency grammar (DG) is a class of modern syntactic theories that are all based on the dependency relation and that can be traced back primarily to the work of Lucien Tesnière. The dependency relation views the (finite) verb as the structural center of all clause structure. All other syntactic units (e.g. words) are either directly or indirectly dependent on the verb. DGs are distinct from phrase structure grammars (constituency grammars), since DGs lack phrasal nodes - although they acknowledge phrases. Structure is determined by the relation between a word (a head) and its dependents. Dependency structures are flatter than constituency structures in part because they lack a finite verb phrase constituent. Dependency is a one-to-one correspondence: for every element (e.g. word or morph) in the sentence, there is exactly one node in the structure of

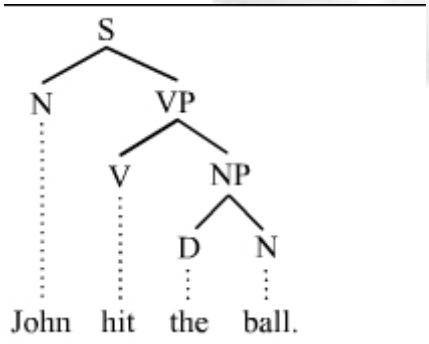
that sentence that corresponds to that element. The result of this one-to-one correspondence is that dependency grammars are word (or morph) grammars. All that exist are the elements and the dependencies that connect the elements into a structure. This situation should be compared with the constituency relation of phrase structure grammars. Constituency is a one-to-one-or-more correspondence, which means that, for every element in a sentence, there are one or more nodes in the structure that correspond to that element. The result of this difference is that dependency structures are minimal compared to their constituency structure counterparts, since they tend to contain many fewer nodes.



Dependency vs Consistency

B. Constituent Reparsing

In constituent reparsing we deal with labeled constituent trees, or phrase structure trees, such as those in the Penn Treebank (after removing traces, empty nodes and function tags). The general idea is the same as with dependencies. First, m parsers each produce one parse tree for an input sentence. We then use these m initial parse trees to guide the application of a parse algorithm to the input. The constituency-based parse trees of constituency grammars distinguish between terminal and non-terminal nodes. The interior nodes are labeled by non-terminal categories of the grammar, while the leaf nodes are labeled by terminal categories. The image below represents a constituency-based parse tree; it shows the syntactic structure of the English sentence John hit the ball:



Constituency based parse tree

The parse tree is the entire structure, starting from S and ending in each of the leaf nodes (John, hit, the, ball). The following abbreviations are used in the tree:

S for sentence; NP for noun phrase; VP for verb phrase, which serves as the predicate;
V for verb; D for determiner; N for noun;

Each node in the tree is either a root node, a branch node, or a leaf node. A root node is a node that doesn't have any branches on top of it. Within a sentence, there is only ever one root node. A branch node is a mother node that connects to two or more daughter nodes. A leaf node, however; is a terminal node that does not dominate other nodes in the tree. S is the root node, NP and VP are branch nodes, and John (N), hit (V), the (D), and ball (N) are all leaf nodes. The leaves are the lexical tokens of the sentence. A node can also be referred to as parent node or a child node. A parent node is one that has at least one other node linked by a branch under it. In the example, S is a parent of both N and VP. A child node is one that has at least one node directly above it to which it is linked by a branch of a tree. From the example, hit is a child node of V. The terms mother and daughter are also sometimes used for this relationship.

Although its roots may be traced back to Panini's grammar of Sanskrit several centuries before the Common Era (Kruijff, 2002) and to medieval theories of grammar (Covington, 1984), dependency grammar has largely developed as a form for syntactic representation used by traditional grammarians, especially in Europe, and particularly in Classical and Slavic domains (Mel'cuk, 1988). This tradition comprises a large and fairly diverse family of grammatical theories and formalisms that share certain basic assumptions about syntactic structure, in particular the assumption that syntactic structure consists of lexical elements linked by binary asymmetrical relations called dependencies. Thus, the common formal property of dependency structures, as compared to representations based on constituency is the lack of phrasal nodes.

III. Dependency Parsing Techniques

A. Graph-Based

A dependency tree is a special case of a dependency graph that spawns from an artificial root and is acyclic. Because of this we can look at a large history of work in graph theory to address finding the best spanning tree for each dependency graph. The most common form of this type of dependency parsing is called arc-factored parsing and deals with the parameterization of the edge weights. The main drawback of these methods is that for non-projective trees, the worst case scenario for most methods is a complexity of O(n3) (Eisner [1996]). However, for non-projective parsing Chu-Liu-Edmond's algorithm has a complexity of O(n2) (McDonald et al. [2005]). The most common tool for doing this is MST parser, which is also used in the noun-phrase bracketing experiments described later in this paper.

B. Transition-Based

Transition-based parsing creates a dependency structure that is parameterized over the transitions used to create a dependency tree. This is closely related to the the shift-reduce constituency parsing algorithms. Due to the notion of picking transitions in an abstract machine, the algorithms

used for these systems tend to be greedy. The benefit of this is that the algorithms have a linear time complexity. However, due to the greedy algorithms, longer arc parses can cause error propagation across each transition (Kubler et al. [2009]). The standard tool for transition-based algorithms is Malt Parser (Nivreet al. [2007b]) which in the shared tasks was often tied with the best performing systems.

C. Constituent Transformation

While not a true dependency parser, one technique often applied is to take a state of the art constituent parser and transform its phrase based output into dependency relations. This has been shown to also be state-of-the-art in accuracy for dependency parsing in English. This method has also been applied to the Czech language with Collin's parser. One path of research should test how this process works in other languages and for treebanks specifically annotated for dependency relations. In most cases the models are built from the Penn Treebank, a constituent based treebank (Marcus et al. [1993]), using a phrase based parser. Then to parse a sentence into a dependency structure, the phrase based output is processed with a conversion tool e.g. Penn Converter (Johansson and Nugues [2007]) or Stanford Converter (Marneffe et al. [2006]). Versions of these converters were used in the CoNLL shared task to create dependency treebanks for a variety of the languages. For my experiments I will make use of Charniak and Johnson [2005] as well as other constituent parsers.

IV. Conclusion

Increasing accuracy of the current parsers should not be the only goal going forward. There are 6,000+ languages in the world and very few of them have dependency trees available. This eliminates the possibility of any supervised training without a very heavy cost in creating the training data from scratch. For this reason, unsupervised methods should be researched and developed further. Current techniques such as Klein and Manning [2004] only get in the 40% - 60% range in undirected and unlabeled accuracy, depending on language and treebank. This is a slight improvement over a baseline that characterizes a dependency as adjacency related. Work in this area is advantageous since the cost of manually creating dependency structures is a bottleneck for most languages in the world and unsupervised methods will give these languages a starting point into the field.

While dependency parsing has made many advancements with the shared task competitions, there is room for improvement to all current models. I hope to primarily focus on annotation standards, domain adaptation, and model combination to improve current performance. Furthermore, I hope to research and improve dependency parsing for under-resourced languages through unsupervised learning.

References

[1] Henderson, J., & Brill, E. (1999). Exploiting diversity in natural language processing: combining parsers. In Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing (EMNLP).

[2] Marcus, M. P., Santorini, B., & Marcinkiewics, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics, 19.

[3] McDonald, R., Pereira, F., Ribarov, K., &Hajic, J. (2005). Non-Projective Dependency Parsing using Spanning Tree Algorithms. In Proceedings of the Conference on Human Language Technologies/Empirical Methods in Natural Language Processing (HLT-EMNLP). Vancouver, Canada.

[4] Nivre, J., & Scholz, M. (2004). Deterministic dependency parsing of English text. In Proceedings of the 20th International Conference on Computational Linguistics (pp. 64-70). Geneva, Switzerland.

[5] Sagae, K., & Lavie, A. (2005). A classifier-based parser with linear run-time complexity. In Proceedings of the Ninth International Workshop on Parsing Technologies. Vancouver, Canada.

[6] Yamada, H., & Matsumoto, Y. (2003). Statistical dependency analysis using support vector machines. In

[7] Proceedings of the Eighth International Workshop on Parsing Technologies. Nancy, France. Zeman, D., & 7. Žabokrtský, Z. (2005). Improving Parsing Accuracy by Combining Diverse Dependency Parsers. In Proceedings of the International Workshop on Parsing Technologies. Vancouver, Canada.

[8] Buchholz, S. and Marsi, E., CoNLL-X shared task on multilingual dependency parsing, in Proceedings of the Tenth Conference on Computational Natural Language Learning, CoNLL-X '06, pp. 149–164, Association for Computational Linguistics, Stroudsburg, PA, USA, URL http://portal.acm.org/citation.cfm?id=1596276.1596305, 2006.

[9] Charniak, E. and Johnson, M., Coarse-to-fine n-best parsing and maxent discriminative reranking, in Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, ACL '05, pp. 173-180, Association for Computational Linguistics, Stroudsburg, PA, USA, URL http://dx.doi.org/10.3115/1219840.1219862, 2005.

[10] Eisner, J., Three new probabilistic models for dependency parsing: An exploration, in Proceedings of the 16th International Conference on Computational Linguistics (COLING-96), pp. 340–345, Copenhagen, URL http://cs.jhu.edu/˜jason/papers/#coling96, 1996.

[11] Green, N., Effects of noun phrase bracketing in dependency parsing and machine translation, in Proceedings of the ACL 2011 Student Session, pp. 69–74, Association for Computational Linguistics, Portland, OR, USA, URL http://www.aclweb.org/anthology/P11-3013, 2011