MaltDiver: A Transition-Based Parser Visualizer

Miguel Ballesteros **Roberto Carlini**
Natural Language Processing Group
Pompeu Fabra University
Barcelona, Spain
{miguel.ballesteros, roberto.carlini}@upf.edu

Abstract

Transition-based dependency parsers are widely used in the Natural Language Processing community but they are normally treated as black boxes, assuming that they provide the dependency parsing of a set of examples. We present *MaltDiver*, a tool developed to visualize the transitions performed by the transition-based parsers included in MaltParser and to show how the parsers interact with the sentences and the data structures within. During the demo session, we will run MaltDiver on several sentences and we will explain the potentialities of such a system.[1]

1 Introduction

Natural language processing researchers apply transition-based parsers frequently, these parsers are implemented in MaltParser (Nivre and Hall, 2005; Nivre et al., 2007b). Most of the application developers make use of the parsers without knowing how these parsers actually work, treating them as black boxes.

In order to have a system that could help to understand how a transition-based parser works, we present *MaltDiver*. MaltDiver is a tool developed to visualize the transitions performed by the transition-based parsers included in MaltParser and to show how they traverse the transition-system. We believe that there are mainly two different target researchers, that belong to different knowledge levels: (i) expert users who are willing to see how the parser behaves with a new set of features or with a different parsing constraint, and (ii) non-expert users who are willing to understand how the parsers work with the sentences that they are interested to parse, helping them to find out errors during the parsing process or inconsistencies in the annotation.

In the rest of the paper, we explain how a transition-based parser works (Section 2), we describe how we have implemented MaltDiver (Section 3), we present related work (Section 4), we show some ideas for further work (Section 5) and we conclude (Section 6).

2 Transition-based parsing - MaltParser

A transition-based parser learns parsing models that are trained to predict the next state of a state machine. To this end, it uses features that are annotated in the input sentence and dependency structure features that are dynamically generated. A typical transition-based parser state, see Figure 1, consists in two data structures (a stack and a buffer), and the partially built dependency structure. The parser starts in an initial state and produces transitions in order to reach new states by using the predictions of the trained model. This kind of parsing is very efficient, normally linear, O(n), in the sentence length and it provides the possibility of using features based on the partially built dependency structure. However, in a transition-based parsing strategy, in which there is a lack of backtracking, it is difficult to avoid an error propagation when it occurs (McDonald and Nivre, 2007). This may serve also as an evidence about why we are interested in the existence of a system as the one that we are presenting in this paper.

MaltParser (Nivre and Hall, 2005; Nivre et al., 2007b) is a transition-based dependency parser generator that provides high results. In the CoNLL Shared Tasks in 2006

[1]The system is available for download at `http://taln.upf.edu/pages/MaltDiver/`. It includes examples and a complete readme file that explains how to use the tool.

The Companion Volume of the Proceedings of IJCNLP 2013: System Demonstrations, pages 25–28,
Nagoya, Japan, 14-18 October 2013.

Nivre's transition system:

Initial configuration → Terminal configuration:

Transitions:

SHIFT: $\langle \Sigma, i|B, H, D\rangle \Rightarrow \langle \Sigma|i, B, H, D\rangle$

REDUCE: $\langle \Sigma|i, B, H, D\rangle \Rightarrow \langle \Sigma, B, H, D\rangle$

LEFT-ARC (r): $\langle \Sigma|i, j|B, H, D\rangle \Rightarrow \langle \Sigma, j|B, H[i \rightarrow j], D[i \rightarrow \mathrm{r}]\}\rangle$
if h(i) $\neq 0$.

RIGHT-ARC (r): $\langle \Sigma|i, j|B, H, D\rangle \Rightarrow \langle \Sigma|i|j, B, H[j \rightarrow i], D[j \rightarrow \mathrm{r}]\}\rangle$
if h(j) $= 0$.

Figure 1: Transition System for arc-eager algorithm.

and 2007 (Buchholz and Marsi, 2006; Nivre et al., 2007a), it was one of the best parsers. MaltParser contains four different families of transition-based parsers, the current version of MaltDiver only handles arc-eager parsing algorithm. These parsers mainly differ in the attachment of right-dependents, being the arc-eager greedier when right attachments have to be generated (Ballesteros and Nivre, 2013).

Figure 1 shows the parsing transitions for Nivre arc-eager with reduce transition: (i) SHIFT, (ii) REDUCE, (iii) LEFT-ARC and (iv) RIGHT-ARC. Nivre's arc-eager parsing algorithm makes use of two data structures in order to handle the input words: a *buffer*, which keeps the words that have to be read, and a *stack*, containing words that have already been processed but they are still available to producing a dependency arc. The SHIFT transition removes the first word in the buffer, and puts it on the top of the stack. The REDUCE transition removes the word that is on the top of the stack because there are no more arcs that have this word as a dependent or as a head. The LEFT-ARC and RIGHT-ARC transitions create either and arc from right to left or left to right, and stores the new arc in the dependency structure H and list D of dependency labels for each word.

3 MaltDiver

MaltDiver is a system implemented in Java that *dives* into the transition-based system with the intention of showing the states that the parser performs for a given sentence. At this writing, our MaltDiver implementation only allows the visualization of the arc-eager parsing algorithm (Nivre, 2003) – but it would not be difficult to include new transition systems (Nivre, 2008) as we also mention in Section 5.

MaltDiver processes the outcome of the *diagnostic feature* of MaltParser,[2] which prints the transition sequence for each sentence of the test corpus. It basically shows the list of transitions and the dependency label selected (if available). Besides that, MaltDiver also makes use of the dependency tree produced in order to ensure the reliability of the transition sequences inferred in the MaltDiver processes.

We included an extra option in MaltDiver which is the one that corresponds with the *allow_root* option in MaltParser.[3] This option decides whether there is a dummy root node included in the first parsing state on the stack. As in MaltParser, the *allow_root* option is set to *true* in default settings.

Therefore, MaltDiver takes the following inputs: (i) input sentence, (ii) a sequence of transitions provided by the MaltParser diagnostic feature and (iii) a dependency tree produced by MaltParser for the input sentence. After that, it processes the list of transitions from left to right and it reconstructs the parser configurations off line.

MaltDiver includes a console version of the system, that prints the parsing states and the dependency structure that is being produced in each state in a pretty-print way. In order to ensure the usefulness of the system, MaltDiver produces a *pdf* file for each state of the parsing process by using the TikZ-dependency tool,[4] which provides a LaTeX interface that we use for the production of the different states and partially built dependency structures. Therefore, the pdf file allows to go backward and forward and save the current state in a separate *pdf* file. Besides the pdf file, the user could also access the LaTeX format file. Figure 2 shows an intermediate state of the pdf file generated within MaltDiver transitions by processing a sentence written in Spanish. The structure to the left of the picture is the **stack**,

[2] This can be achieved by using the following setting in MaltParser: *-di true -dif filename.log*

[3] See `www.maltparser.org/userguide.html`

[4] TikZ-dependency tool is available for downloading through `https://sourceforge.net/projects/tikz-dependency/`

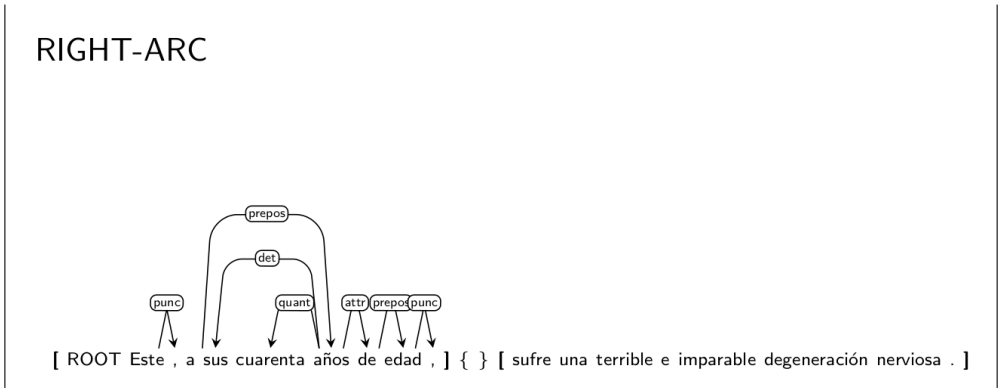

Figure 2: A print-screen of the system for the following sentence written in Spanish: *Este, a sus cuarenta años de edad, sufre una terrible e imparable degeneración nerviosa.* The structure to the left of the picture is the **stack**, and the one to the right is the **buffer**.

and the one to the right is the **buffer**.

4 Related Work

The importance of visualization systems has been evidenced during the last years in the NLP community. In the parsing and generation area we can find systems, such as MaltEval (Nilsson and Nivre, 2008), the Mate Tools (Bohnet et al., 2000), XLDD (Culy et al., 2011) or more recently, TreeExplorer (Thiele et al., 2013), which are, among other things, systems that visualize parse trees for evaluation and to provide the option of exploring dependency structures.

We also consider relevant and motivated in a similar way the work developed by Christopher Collins et al. about visualization of linguistic data in the Computer Graphics area (Collins et al., 2009a; Collins et al., 2009b), in which they present interactive visualization systems for NLP in discourse analysis, document content and even machine translation parse trees.

5 Future Work

A tool like MaltDiver provides several future directions and applications in different scenarios. The first idea would be to include other parsers in the system, such as the ones included in MaltParser that are not treated with MaltDiver. Some of them would be very easy to include, because they share with Nivre's parsers the transition system. However, there are some parsers that are a challenge, because we would have to include additional data structures in the visualization.

We could also provide an implementation of the pseudo-projective transformation of Nivre and Nilsson (2005) in the system process. We believe that the implementation of this step is rather straightforward, because we would only have to trace the projective parsing process –as we have already done– resulting in the pseudo-projective tree before post-processing. By comparing this to the final tree output by the system, we can then infer which arcs were moved due to pseudo-projective parsing. In fact, this is something that an user could do manually in the current version of MaltDiver.

A great idea would be to integrate MaltDiver with MaltOptimizer (Ballesteros and Nivre, 2012) in order to understand how the parser changes its behavior by updating the features selected.

6 Conclusions

We have presented MaltDiver, a tool that may serve as support for people interested in parsing research. This kind of tool would allow to understand the parsing processing by preparing resources about transition-based parsing in short time; researchers with deep knowledge about transition-based parsing could find it useful in order to understand the outcomes that the parsers may produce for a given sentence. For instance, a possible MaltDiver use could be an automatic comparison between different parsing behaviors for experiments about parsing root positions (Ballesteros and Nivre, 2013) or parsing directions modifications (Attardi and Dell'Orletta, 2009) and (Hall et al., 2007).

Acknowledgments

Thanks to Joakim Nivre, Johan Hall and Leo Wanner for their kind support and useful comments.

References

Giuseppe Attardi and Felice Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 261–264.

Miguel Ballesteros and Joakim Nivre. 2012. MaltOptimizer: A System for MaltParser Optimization. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*.

Miguel Ballesteros and Joakim Nivre. 2013. Going to the roots of dependency parsing. *Computational Linguistics*, 39(1):5–13.

Bernd Bohnet, Andreas Langjahr, and Leo Wanner. 2000. A development environment for an mtt-based sentence generator. In *Proceedings of the First International Natural Language Generation Conference*.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.

Christopher Collins, M. Sheelagh T. Carpendale, and Gerald Penn. 2009a. Docuburst: Visualizing document content using language structure. *Comput. Graph. Forum*, 28(3):1039–1046.

Christopher Collins, Gerald Penn, and M. Sheelagh T. Carpendale. 2009b. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1009–1016.

Chris Culy, Verena Lyding, and Henrik Dittmann. 2011. xldd: Extended linguistic dependency diagrams. In *Proceedings of the 2011 15th International Conference on Information Visualisation*, IV '11, pages 164–169, Washington, DC, USA. IEEE Computer Society.

Johan Hall, Jens Nilsson, Joakim Nivre, Gülsen Eryiğit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 933–939.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131.

Jens Nilsson and Joakim Nivre. 2008. Malteval: an evaluation and visualization tool for dependency parsing. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may. European Language Resources Association (ELRA).

Joakim Nivre and Johan Hall. 2005. MaltParser: A language-independent system for data-driven dependency parsing. In *Proceedings of the 4th Workshop on Treebanks and Linguistic Theories (TLT)*, pages 137–148.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007a. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryiğit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007b. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13:95–135.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.

Gregor Thiele, Markus Gärtner, Wolfgang Seeker, Anders Björkelund, and Jonas Kuhn. 2013. Treeexplorer – an extensible graphical search tool for dependency treebanks. In *Proceedings of the Demonstrations of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*.