Palmyra: A Platform Independent Dependency Annotation Tool for Morphologically Rich Languages

Talha Javed, Nizar Habash, Dima Taji
Computational Approaches to Modeling Language Lab
New York University Abu Dhabi
{talha.javed, nizar.habash, dima.taji}@nyu.edu

Abstract

We present Palmyra, a platform independent graphical dependency tree visualization and editing software. Palmyra has been specifically designed to support the complexities of syntactic annotation of morphologically rich languages, especially regarding easy change of morphological tokenization through edits, additions, deletions, splits and merges of words. Palmyra uses an intuitive drag-and-drop interface for editing tree structures, and provides pop-up boxes and keyboard shortcuts for part-of-speech and link label tagging.

Keywords: Annotation, Interfaces, Syntax, Morphologically Rich Languages

1. Introduction

Producing high-quality natural language syntactic annotation is expensive. Well-known large-scale syntactic annotation projects, such as the Penn Treebank (Marcus et al., 1993) and the Prague Dependency Treebank (Böhmová et al., 2003), relied on expert linguists to produce carefully annotated data. This process is rather costly, and as a result, such annotation projects have been undertaken for only a handful of important languages.

Efficient annotation tools play an important role in lowering treebank development costs and enabling the creation of larger, higher quality treebanks. The typical approach is to automatically create syntactic annotations, which are then manually corrected. In this scenario, a goal of the annotation tool is to lower the annotation burden on the annotators as much as possible. For morphologically rich languages (MRLs), such as Arabic and Hebrew, syntactic annotation includes morphological ambiguity resolution as well as tokenization adjustment. The change of tokenization affects the total token count of a sentence and requires adding or deleting tree tokens. For example, the word وجدها *wjdhA*[1] may be automatically analyzed as وجد+ها *wjd*/VERB+*hA*/PRON 'he found her', but needs to be corrected to و+جد+ها *w*/CONJ+*jd*/NOUN+*hA*/PRON[2] 'and her grandfather' (or vice versa). Since an automatic parser may select an incorrect analysis and tokenization that will need to be corrected, an interface that facilitates such corrections during syntactic annotation would be most helpful.

Palmyra is an annotation tool intended to help with the annotation of MRLs in general. However, we focus on Arabic here as a representative language for the type of challenges we are interested in addressing. We are primarily interested in dependency treebanking and thus the discussion of related work will be focused on dependency representations.

[1]Arabic transliteration is presented in the Habash-Soudi-Buckwalter scheme (Habash et al., 2007).

[2]For more information on Arabic morphology and natural language processing, see (Habash, 2010).

2. Design Aims

The following are the design aims for the Palmyra editor interface. **First**, we want the editor to provide an easy and direct way to modify the form of a word: by changing it, splitting it, or merging it with other words. **Second**, we want the editor to allow easy switching between syntactic annotation and morphological tokenization to allow the annotators to make these decisions jointly and avoid error propagation. **Third**, we want the editor to be platform independent, and require no installation effort. **Fourth**, we want the editor to be language-independent and easily configurable. **Finally**, we want the editor to be open source so it can be easily extended for other projects without restrictions.

3. Related Work

Much work has been done on improving dependency parsing and thus many tools for annotation have been created. The most famous and commonly used tool is TrEd. BRAT and WebAnno are web-based tools for text annotation, designed mainly for collaboration. EasyTree is another lightweight tool designed to be run in browsers and consists of only a front-end. These tools have some great features but also lack key features which do not make them completely suitable for annotating MRLs. In particular, changing the tokenization of a word is not easy to do in these systems. We discuss these tools next to highlight their advantages and shortcomings.

TrEd (Pajas and Štěpánek, 2008) is a graph visualization and manipulation program written in Perl. It has been used as an annotation tool for several treebank projects, including the Prague Dependency Treebank (Böhmová et al., 2003), Prague Arabic Dependency Treebank (Hajic et al., 2004) and Columbia Arabic Treebank (CATiB) (Habash and Roth, 2009). It supports macros to automate frequently repeated operations and has a substantial number of features. It can be unintuitive at times and difficult to learn; thus, it may not be a good choice for less experienced annotators. A notable limitation of TrEd is that it is a standalone application and thus cannot be run through a web-browser. Palmyra is designed to run in web browsers, making it simple to use and appropriate for web-based annotation tasks. TrEd does not provide any simple option for word tokenization.

BRAT (Stenetorp et al., 2012) is a web-based annotation tool with a focus on collaborative annotation. BRAT is designed in particular for structured annotation, where the notes are not freeform text but have a fixed form that can be automatically processed and interpreted by a computer. The way BRAT displays text, on a single line, makes following the dependency arcs somewhat difficult, and thus, this tool is probably more appropriate for other tasks, such as marking events and named entities. Furthermore, BRAT is slow when processing documents of more than 100 sentences and has limited support for different file formats. It also does not allow for web-based configuration of tag sets.

WebAnno (Yimam et al., 2013) is also another general-purpose web-based annotation tool mainly meant for distributed teams. To visualize the text and annotations, it uses the JavaScript based annotation visualization from BRAT, thus facing the same display issues. WebAnno supports type specification through the import/export of tag sets. Similar to TrEd, WebAnno and BRAT have no support for word tokenization.

EasyTree (Little and Tratz, 2016) is a light-weight tool designed for annotating dependency trees in browsers. It is limited to a front-end only and does not provide any interface to integrate parsers for pre-annotation. EasyTree has multiple intuitive features, such as color-coded part-of-speech (POS) indicators and optional translation displays. EasyTree allows the customization of POS tags but does not maintain sentence order of nodes. It has no functionality for splitting or merging of words. Palmyra's design is highly influenced by EasyTree and uses some of its code.

4. Palmyra Design Specifications

Design and Implementation Palmyra was written entirely in JavaScript, CSS, and HTML. As such it can run in modern web browsers and is platform independent and requires no special installation process.[3] For the interactive graphical display, Palmyra leverages D3, a popular open source data visualization library written in JavaScript. Palmyra utilizes UTF-8 encoding, which enables it to work with characters from different languages. Palmyra is built on top of the base code of EasyTree (Little and Tratz, 2016) and makes use of its drag-and-drop functionality. Figure 2 shows the main Palmyra interface.

Input Files For input, Palmyra supports a basic dependency file format containing five columns; ID, word, POS, parent, and relation. Palmyra input files may contain a number of independent trees, which can be browsed through. Editing is performed on one tree at a time. Figure 1 shows the five columns used to represent the dependency tree shown in Figures 2 and 4. The columns, in order, correspond to (i) a unique word index, (ii) the word, (iii) the CATiB POS tag (Habash and Roth, 2009), (iv) the word's parent, (v) the relation (link) label. The words in this tree

ID	Word	POS	Parent	Relation
1	و+	PRT	3	MOD
2	لم	PRT	3	MOD
3	يتم	VRB	0	---
4	اعتقال	NOM	3	SBJ
5	أحد	NOM	4	IDF
6	ب+	PRT	3	MOD
7	حسب	NOM	6	OBJ
8	الشرطة	NOM	7	IDF
9	.	PNX	3	MOD

Figure 1: The tree for the sentence ولم يتم اعتقال أحد بحسب الشرطة *wlm ytm AςtqAl ÂHd bHsb AlšrTħ* 'and no one was arrested according to the police."

are tokenized according to the Penn Arabic Treebank tokenization scheme (Maamouri et al., 2004), but other Arabic tokenization schemes can be used just the same (Habash, 2010). The format shown is produced by CamelParser (Shahrour et al., 2016).

Palmyra also accepts sentences for input when annotating from scratch. Every input line is considered as a separate tree. It tokenizes a sentence on spaces, and assumes all nodes are siblings under the root with default POS tag and link labels.

Output Files Palmyra has two output formats: the dependency format discussed above, and PNG image.

Configuration File Palmyra takes as input an optional configuration file, which specifies the various configuration and annotation options for the tool. The config files consist of key-value pairs, each specifying a property of the editor. The most important use of the config file is to specify the options for POS tags, link labels, and keyboard shortcuts.

5. Tree Node Editing

Dependency Tree Display Figure 2 presents a screenshot of a dependency tree in Palmyra. The words are shown at the bottom in a straight horizontal line, and the tree nodes are aligned horizontally directly above the corresponding words, and vertically according to the structure of the tree. Palmyra leverages D3-hierarchy module which includes layout algorithms for visualizing hierarchical data. The POS of a word is displayed next to it, and the relation link is shown half-way on the edge connecting two tree nodes. The direction of display of the tree can be changed to be left-to-right or right-to-left.

Drag-and-Drop and Zoom Tree editing in Palmyra is straightforward; users simply click on word nodes and move them around using drag-and-drop. When the user begins dragging a node, red circles appear around the remaining nodes. These 'drop zones' indicate where the node may be re-attached. Figure 2 displays the tree during editing, when the node الشرطة is being dragged. When the dragged node is eventually dropped onto a drop zone, a link representing a syntactic dependency is created between the two nodes with the dropped node as the child. The re-attached

[3]Try Palmyra online at `https://camel.abudhabi.nyu.edu/palmyra/`.

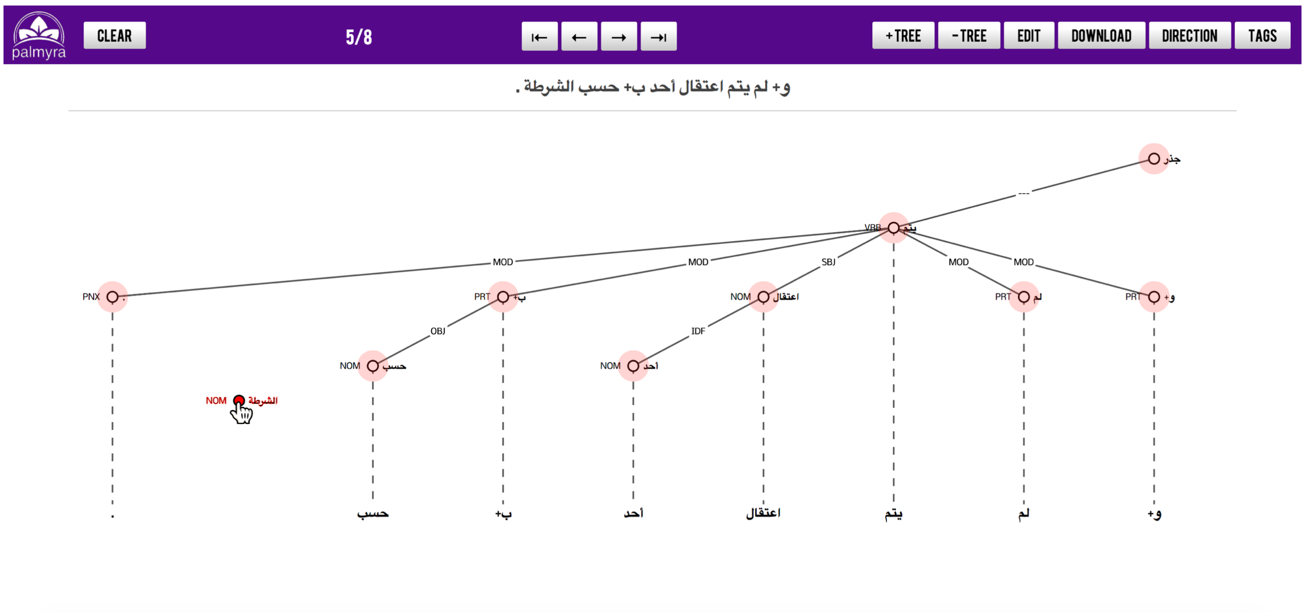

Figure 2: PALMYRA Interface while editing a tree using the drag-and-drop functionality

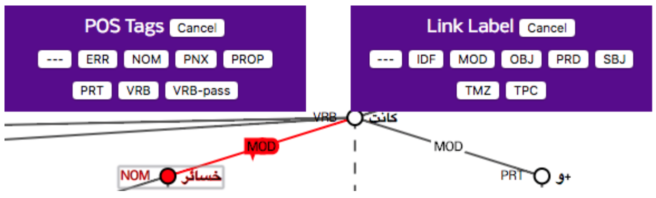

Figure 3: POS Tag and Link Label Selection

node is inserted such that it and its new siblings remain sorted according to the original word order of the sentence. The tool also includes tree zoom-in and zoom-out options.

Keyboard Tree Node Navigation Tree node selection can be done by directly clicking on the node of interest. But PALMYRA also allows the user to move from one node to another using keyboard arrows. Node-highlighting provides feedback on the selected node and facilitates fast directed movement within the tree: to go up or down a parent connection or jump to siblings' subtrees using keyboard arrows.

Part-of-Speech Tag and Link Label Clicking on the "TAGS" button opens two sub-menus with the available options for POS tags and link labels (Figure 3). The user must select a node before being able to change the POS tag or link label. A node can be selected by either clicking on the node or navigating to it through keyboard arrow keys. Once selected, the user can change the POS tag or link label either through keyboard shortcuts or by clicking on the displayed buttons in the TAGS sub-menus using a mouse.

Adding and Deleting Trees The user can also add a new tree to the file using the "+ TREE" button. The tree starts with only a root, and the user can use the word insertion functionality (discussed in the next section) to add nodes to the tree. The "- TREE" button can be used to delete a tree.

6. Sentence Token Editing

The sentence token editing mode is toggled when clicking the "EDIT" mode button in the upper interface bar. Within this mode, the annotator can delete, insert, change, split or merge tree nodes. Figure 4 displays the tree in sentence token editing mode.

Deletion In the "EDIT" mode, PALMYRA displays a red "x" sign under nodes, which can be clicked to delete nodes. If the deleted node has children, they are all promoted to be the children of the deleted node's parent.

Insertion Similarly, a green "+" sign is shown between nodes to provide the ability to add a new node in the respective position. Inserted nodes are linked to the root and given the word form New/جديد.

Substitution and Word Splitting Clicking the word in edit mode causes a word edit pop-up to appear. The word can be rewritten for a simple substitution. A word can also be split into multiple words that will then be represented as different nodes in the tree. This is done by simply adding spaces in the middle of the word where the splits should take place.

Merge Words Merging neighboring words can be accomplished by clicking on the arrows to the right or left of a word. The node for whom the merge arrows are clicked is deleted and its name (word) is added to the name (word) of the respective neighbor node (on left or right given the clicked arrow). The original node loses its POS tag and link

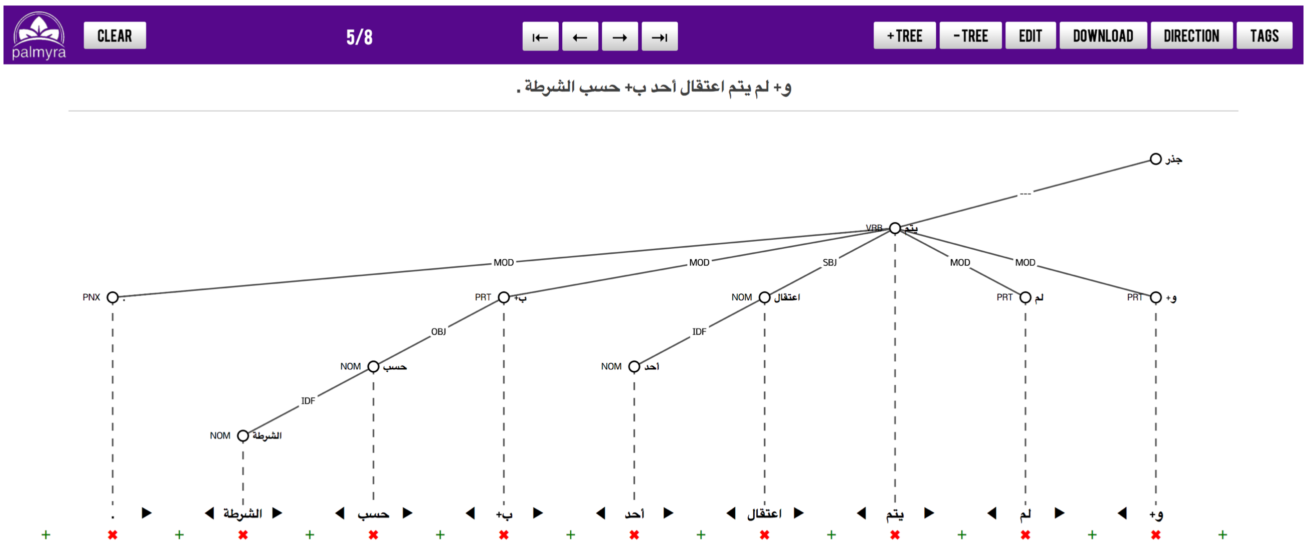

Figure 4: PALMYRA Interface in sentence token editing mode

label, and its children are assigned to the node it merged with. At the end, only one node remains with its word being the concatenation of the words of the two original nodes.

7. Conclusion and Future Work

In this paper, we described PALMYRA, a platform independent graphical dependency tree visualization and editing software. PALMYRA has been specifically designed to support the complexities of syntactic annotation of morphologically rich languages, especially regarding easy change of morphological tokenization. Being built entirely using standard web technologies, PALMYRA runs on all major web browsers and is ideal for online annotation efforts, such as crowdsourcing efforts.

In the future, we will be using PALMYRA heavily as part of a treebanking annotation project. We plan to continue enhancing PALMYRA's capabilities. In particular, we are interested in linking it to a state-of-the-art Arabic parser (Shahrour et al., 2016) to support online syntactic analysis.

8. Bibliographical References

Böhmová, A., Hajič, J., Hajičová, E., and Hladká, B. (2003). The Prague dependency treebank. In *Treebanks*, pages 103–127. Springer.

Habash, N. and Roth, R. M. (2009). Catib: The Columbia Arabic treebank. In *Proceedings of the ACL-IJCNLP*, Stroudsburg, PA, USA.

Habash, N., Soudi, A., and Buckwalter, T. (2007). On Arabic Transliteration. In A. van den Bosch et al., editors, *Arabic Computational Morphology: Knowledge-based and Empirical Methods*. Springer.

Habash, N. (2010). *Introduction to Arabic Natural Language Processing*. Morgan & Claypool Publishers.

Hajic, J., Smrz, O., Zemánek, P., Šnaidauf, J., and Beška, E. (2004). Prague Arabic dependency treebank: Development in data and tools. In *Proc. of the NEMLAR Intern. Conf. on Arabic Language Resources and Tools*, pages 110–117.

Little, A. and Tratz, S. (2016). Easytree: A graphical tool for dependency tree annotation. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, 2016.*

Maamouri, M., Bies, A., Buckwalter, T., and Mekki, W. (2004). The Penn Arabic Treebank: Building a Large-Scale Annotated Arabic Corpus. In *NEMLAR Conference on Arabic Language Resources and Tools*, pages 102–109, Cairo, Egypt.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330.

Pajas, P. and Štěpánek, J. (2008). Recent advances in a feature-rich framework for treebank annotation. In *Proceedings of the 22Nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 673–680, Stroudsburg, PA, USA. Association for Computational Linguistics.

Shahrour, A., Khalifa, S., Taji, D., and Habash, N. (2016). Camelparser: A system for Arabic syntactic analysis and morphological disambiguation. In *COLING (Demos)*, pages 228–232.

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., and Tsujii, J. (2012). Brat: A web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pages 102–107, Stroudsburg, PA, USA.

Yimam, S. M., Gurevych, I., Eckart de Castilho, R., and Biemann, C. (2013). Webanno: A flexible, web-based and visually supported system for distributed annotations. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 1–6, Sofia, Bulgaria.