A Web-based System for Crowd-in-the-Loop Dependency Treebanking

Stephen Tratz[*], **Nhien Phan**[†]
[*]U.S. Army Research Laboratory, Adelphi, Maryland, USA
[†]University of Maryland, College Park, Maryland, USA
stephen.c.tratz.civ@mail.mil, nphan12@terpmail.umd.edu

Abstract

Treebanks exist for many different languages, but they are often quite limited in terms of size, genre, and topic coverage. It is difficult to expand these treebanks or to develop new ones in part because manual annotation is time-consuming and expensive. Human-in-the-loop methods that leverage machine learning algorithms during the annotation process are one set of techniques that could be employed to accelerate annotation of large numbers of sentences. Additionally, crowdsourcing could be used to hire a large number of annotators at relatively low cost. Currently, there are few treebanking tools available that support either human-in-the-loop methods or crowdsourcing. To address this, we introduce CROWDTREE, a web-based interactive tool for editing dependency trees. In addition to the visual frontend, the system has a Java servlet that can train a parsing model during the annotation process. This parsing model can then be applied to sentences as they are requested by annotators so that, instead of annotating sentences from scratch, annotators need only to edit the model's predictions, potentially resulting in significant time savings. Multiple annotators can work simultaneously, and the system is even designed to be compatible with Mechanical Turk. Thus, CROWDTREE supports not simply human-in-the-loop treebanking, but crowd-in-the-loop treebanking.

Keywords: annotation, dependency trees, visualization, GUI, crowdsourcing, treebanking

1. Introduction

Although treebanks exist for a number of languages, they are often quite small in size. Even some of the largest and most useful treebanks are restricted to only a handful of sources, genres, and/or topic areas—the venerable Penn Treebank (Marcus et al., 1993) being a prime example of this. Unfortunately, manual annotation is expensive and slow, which impedes both the expansion of existing treebanks and the creation of new ones. For example, The Prague Dependency Treebank, which contains over 1 million syntactically connected words, cost approximately $600,000 to create (Böhmová et al., 2003). One method for accelerating the treebanking process is to utilize human-in-the-loop[1] methods that leverage machine learning algorithms during annotation. Such algorithms could pre-parse the data, highlight attachment decisions of greater importance, and/or suggest sentences for annotation that are most likely to prove useful as training examples. Another useful method is crowdsourcing, which taps into a large (and typically inexpensive) worker pool. Together, these complementary techniques hold tremendous promise for reducing the time and cost of creating large datasets for natural language processing, including treebanks. In order to facilitate faster and less expensive dependency tree annotation, this paper presents CROWDTREE, which is, to the best of our knowledge, the first web-based treebanking tool designed specifically for human-in-the-loop, and even crowd-in-the-loop, tree annotation.

The system consists of two primary components, an interactive graphical frontend, shown in Figure 1, and a Java servlet. The frontend, which enables users to visually construct/edit dependency trees using drag-and-drop, is based on the EASYTREE dependency tree editor (Little and Tratz, 2016) but has a variety of improvements, including an improved layout and various bug fixes. It is written entirely in HTML, CSS, and JavaScript, and, thus, works with modern browsers without the need for special plugins. The frontend communicates with the second component, the Java servlet, which, in addition to recording the annotated data, has the option to constantly train a transition-style parsing model (McDonald and Nivre, 2007) throughout the annotation process from the work that is submitted to the server. This model can parse a sentence as it is provided to an annotator, which promises to reduce the number of edits required to construct correct dependency trees. CROWDTREE has a Mechanical Turk-compatible mode, which opens new possibilities for large scale treebanking as well as new opportunities for research into the use of large numbers of non-expert treebankers.

The remainder of this paper provides a more detailed description of the nature and capabilities of our tree annotation system (Section 2), describes related tools and research efforts (Section 3), summarizes our contributions (Section 4), and details some of our planned experimental work and feature additions (Section 5). We are planning to release our system open source in the near future via the U.S. Army Research Laboratory's GitHub site or similar repository.

2. System Description

CROWDTREE has a client-server architecture with two components: a graphical web-based interface that annotators use to create dependency trees and a Java servlet for storing user annotations and running machine learning processes.

2.1. User Interface

The user interface, shown in Figure 1, is written using standard web technologies—HTML, CSS, and JavaScript. The interactive Support Vector Graphics-based dependency tree widget is implemented using the popular D3.JS[2] data

[1]By 'human-in-the-loop', we mean 'involving both human interaction and machine learning'.

[2]https://d3js.org/

visualization and manipulation JavaScript library. Anno-

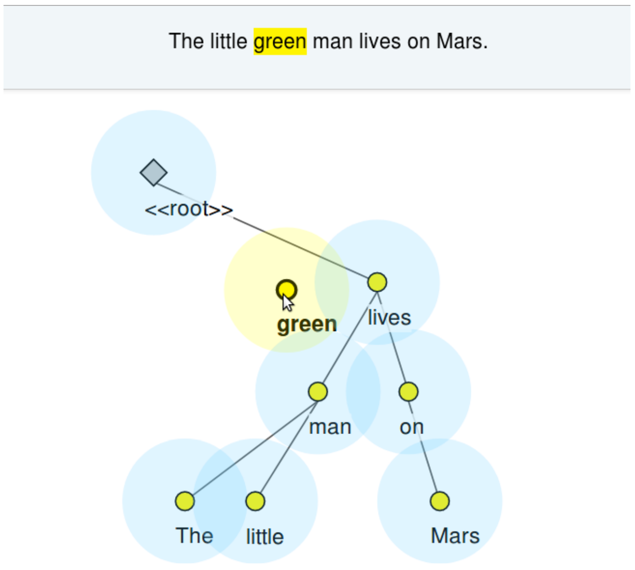
The little green man lives on Mars.

Figure 1: In CROWDTREE, blue "drop zones" highlight possible attachment sites when dragging word nodes.

tators alter the dependency tree structure by clicking on word nodes and dragging them to their appropriate attachment sites. When the dragging process begins, circular "drop zones" appear around the potential attachment sites, as shown in Figure 1. When the dragged node is dropped, the layout of the tree adjusts in a smooth, animated fashion. The code is derived from the EASYTREE dependency tree editor (Little and Tratz, 2016) but the visual layout is modeled after the layout used in the tree editor TRED (Pajas and Štěpánek, 2008), with the words maintaining their original left-to-right order on the horizontal axis. One of the challenges with the EASYTREE layout is that the horizontal ordering of word nodes is not maintained across the entire sentence, and, hence, a noun phrase like *the little brown flower pot* would have the last two words *brown* and *flower* appear to the right of the word *pot* instead of to its left.

Both labeled and unlabeled dependency annotation are supported by the system, which can be configured to either hide or show the dependency labels and part-of-speech tags of the word nodes. Plugging in one's own dependency labels and part-of-speech tags can be accomplished easily with just a text editor.

CROWDTREE includes pan and zoom capabilities to enable users to easily work with very large trees. Zooming in and out is accomplished using the mouse scrollwheel, and the user can reposition the entire tree by clicking on the background and dragging the mouse in the desired direction.

2.2. Java Servlet

The Java servlet, in addition to handling some important input and output functions, is in charge of running machine learning processes. It currently has the option of training either a SWAP parser (Nivre, 2009) or an EASY-FIRST style parsing model (Goldberg and Elhadad, 2010) with support for non-projectivity via swapping (Tratz and Hovy, 2011). The servlet runs a model training thread that continuously iterates over the annotations that have been received; thus, the model tends to improve as annotation proceeds. Whenever an annotator requests a sentence from the servlet, the parser processes the sentence using its current model weights and sends the resulting parse to the annotator for correction. The same parsing model is shared across all annotators, which enables them to benefit from each other's work. This promises to accelerate the entire annotation process by minimizing the number of edits annotators need to make in order to construct correct parse structures. [3]

2.3. Input and Output

One of the significant limitations of the EASYTREE dependency tree editor, which our system is derived from, is that the default method for getting data into and out of the editor is to copy and paste an individual sentence, make any needed changes, and then save out a file containing the annotation. This process is, of course, rather tedious, not to mention disruptive to the annotator's concentration. With our tree editor, the locations of the input and output files are specified in a system configuration file. The sentences contained in the input file are loaded by the Java servlet and are sent, as requested, to the web client via HTTP/HTTPS in JavaScript Object Notation (JSON). The annotations produced by the annotator(s) are then returned to the servlet in a similar fashion. When the servlet receives annotated trees, it writes them out to the file specified in the configuration and adds them to the pool of training instances that the server-side parser learns from.

CROWDTREE includes Java classes for reading files in different input formats, including plain text, CoNLL-U, and our own format. To support as many formats as possible, it is also possible to implement a custom reader Java class and plug it in. The frontend is capable of displaying both left-to-right and right-to-left scripts, and UTF-8 encoding is used throughout the system, so it can support a wide variety of different languages.

2.4. Mechanical Turk Mode

One of the most notable features of CROWDTREE is that it was designed from the beginning to be able to run in connection with the Amazon Mechanical Turk crowdsourcing platform. This is accomplished via Mechanical Turk's *ExternalQuestion* Human Intelligence Tasks (HITs). HITs are the basic unit of work on Mechanical Turk; essentially, they are questions to be answered, with a monetary award attached. In the case of *ExternalQuestion* HITs, the requester provides Amazon with the URL of a web site located externally to the main Mechanical Turk website. As shown in Figure 2, when the workers on Mechanical Turk view and work on the HITs, they see a web page hosted on the main Mechanical Turk website with an internal frame (an HTML *iframe*) that points to the external website hosting the CROWDTREE system. When the worker submits his/her work, it is sent both to the external server, where it can be

[3]Of course, submission of erroneous annotations will likely degrade the performance of the model. Mechanisms for mitigating this negative effect may be necessary, depending on the use case.

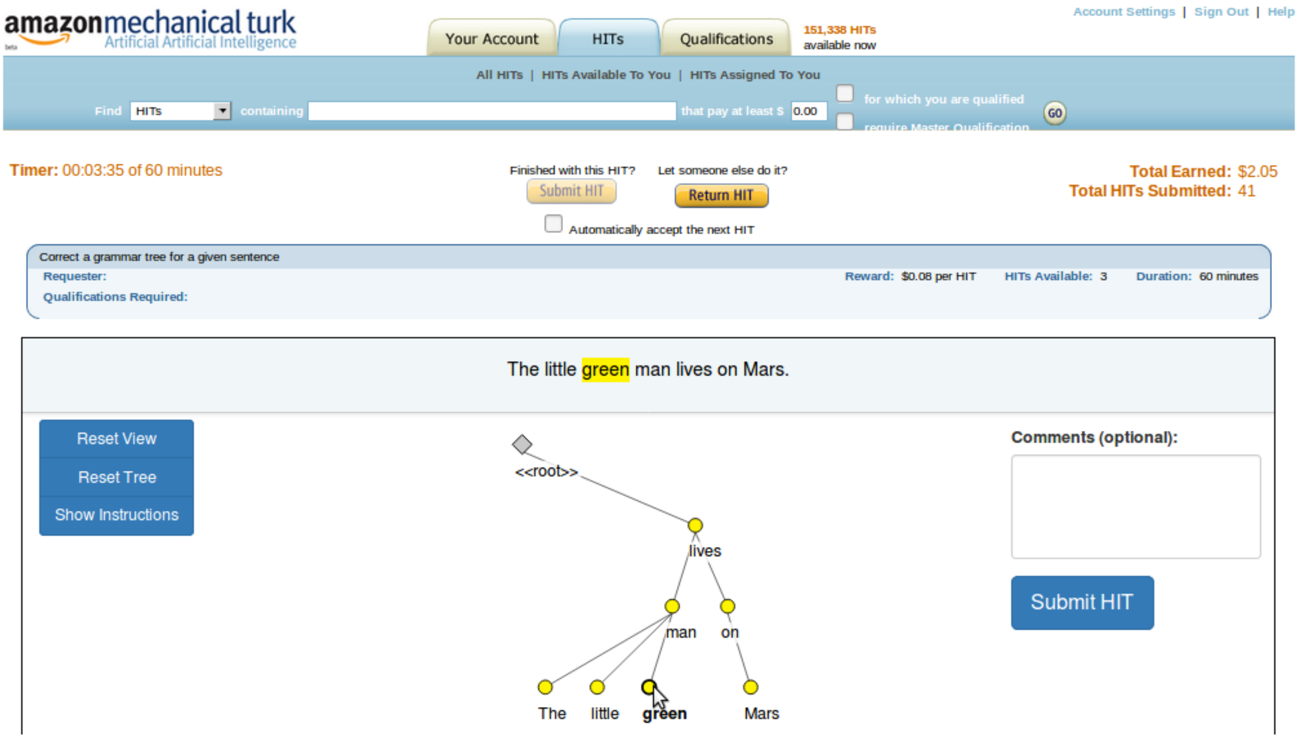
amazonmechanical turk
Artificial Artificial Intelligence
Your Account
HITs
Qualifications
151,338 HITs available now
Account Settings | Sign Out | Help
All HITs | HITs Available To You | HITs Assigned To You
Find HITs containing that pay at least $ 0.00 for which you are qualified require Master Qualification GO
Timer: 00:03:35 of 60 minutes
Finished with this HIT? Submit HIT
Let someone else do it? Return HIT
Automatically accept the next HIT
Total Earned: $2.05
Total HITs Submitted: 41
Correct a grammar tree for a given sentence
Requester:
Qualifications Required:
Reward: $0.08 per HIT
HITs Available: 3
Duration: 60 minutes
The little green man lives on Mars.
Reset View
Reset Tree
Show Instructions
<<root>>
lives
man
on
The
little
green
Mars
Comments (optional):
Submit HIT

Figure 2: Screenshot of CROWDTREE working in the Mechanical Turk sandbox.

accessed by the parsing model training thread (or any machine learning processes implemented in the future), and to the Mechanical Turk website so that the worker can proceed to the next HIT, have his/her work approved, and receive payment.

2.5. Standalone Mode

As mentioned above, CROWDTREE was designed with Mechanical Turk in mind. It is possible, however, for annotators to connect directly to the server hosting the Java servlet instead. In this use case, annotators work on each sentence in the order that they appear in the input file. In the future, we would like to add more sophisticated methods for assigning particular sentences to specific annotators as well as tools that would help supervisors review annotators' work.

3. Related Work

3.1. Treebanking Tools

Perhaps the best known tool for treebanking is the tree editor TREd (Pajas and Štěpánek, 2008). TREd is a programmable graphical user interface for editing and viewing parse trees, including both constituent and dependency trees, and has been used for several treebanking projects, including the Prague Arabic Dependency Treebank (Hajič et al., 2004). It has a wide array of functionality—more than our system—but can be difficult to set up and learn. One of TREd's greatest limitations is that it is a standalone application written in Perl. In contrast, CROWDTREE is designed to run in modern web browsers, making it comparatively simple to deploy, especially for geographically distributed annotators. Some aspects of the visual layout and styling of CROWDTREE were inspired by TREd.

Another web-based tool that can be used for treebanking is WEBANNO (Yimam et al., 2013; Yimam et al., 2014; de Castilho et al., 2016). It builds upon the BRAT interface (Stenetorp et al., 2012) and supports multiple layers of annotation across a variety of linguistic annotation tasks. Additionally, WEBANNO is designed to support annotation across geographically distributed sites and has an integrated machine learning model for suggesting span annotations. It may, however, be more appropriate for semantic labeling tasks, such as word sense disambiguation (WSD) and semantic role labeling (SRL), than for syntactic annotation; although compact, we hypothesize that its horizontal word layout, an example of which is depicted in Figure 3, is slower for treebanking and more exhausting to annotators because they have to follow curved arcs with their eyes and pay attention to the arrowheads at the ends in order to understand the directionality of the dependencies.
Three other dependency tree editors with horizontal layouts similar to WEBANNO but designed with a more focused range of functionality are ARBORATOR (Gerdes, 2013), UD ANNOTATRIX (Tyers et al., 2018), and DGANNOTATOR[4].

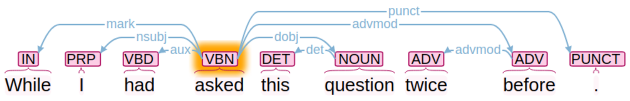

Figure 3: Screenshot showing the type of horizontal layout used in BRAT, WEBANNO, and most other dependency annotation tools.

[4]http://medialab.di.unipi.it/Project/QA/Parser/DgAnnotator/

3.2. Crowdsourcing Parsing

To date, there have been very few efforts to crowdsource parsing and no efforts, to the best of our knowledge, to do so directly with a full parse tree editor like CROWDTREE outside small classroom studies like that of Gerdes (2013). In one notable ongoing effort, researchers have built and deployed a Game With a Purpose (GWAP) (Von Ahn, 2006) called ZOMBILINGO in order to crowdsource a French treebank (Fort et al., 2014; Guillaume et al., 2016; Fort et al., 2017). ZOMBILINGO participants work on one attachment decision at a time, using the metaphor that the governing word is devouring the child just as zombies seek out brains. Participants must complete a training phase for each dependency relation they work on and can not work on the relations deemed more difficult until they have demonstrated some skill on less challenging dependency relations. ZOMBILINGO relies entirely on volunteers and is a standalone website[5] not designed for integration with Mechanical Turk or similar crowdsourcing platforms.

Another notable effort is that of He et al. (2016), who, in a first step toward human-in-the-loop parsing, crowdsource individual attachment annotations by asking annotators multiple choice questions automatically generated using parse trees produced by an existing parser. For example, to resolve the attachment of a relative clause with a root verb of *sells*, the question posed to the annotators would be something like, "What *sells* something?", and the available answers would be the list of noun phrases that appear prior to the relative clause. He et al. are able to achieve some modest gains, improving by 0.2 F1 on their in-domain corpus and 0.6 F1 on their out-of-domain corpus.

3.3. Crowdsourcing combined with Machine Learning

Even though using crowdsourcing to collect data for processing with machine learning algorithms has been widespread for a number of years, it is still relatively rare to run machine learning algorithms at the same time that the crowd is actively annotating. This is especially true in the field of natural language processing (NLP). One notable exception to this rule is the work of Laws et al. (2011), who ran active learning processes while crowdsourcing Named Entity Recognition (NER) and sentiment detection tasks on Mechanical Turk. To the best of our knowledge, no one has utilized machine learning in any similar fashion while crowdsourcing parse trees.

4. Conclusion

In this paper, we introduce CROWDTREE, an interactive graphical editor for dependency tree annotation. Based upon standard web technologies, the system is compatible with modern browsers and, thus, easy to deploy to geographically distributed annotators. Our system's integrated back end parsing model makes it suitable for human-in-the-loop or even crowd-in-the-loop annotation, where the integrated parsing model learns as the annotation proceeds so that, ideally, the amount of effort required per sentence decreases as time goes on. CROWDTREE supports left-to-right and right-to-left scripts, labeled and unlabeled dependency annotation, and can run in conjunction with Mechanical Turk (via *ExternalQuestion* HITs) or in standalone mode. To the best of our knowledge, it is the first dependency tree editor designed for use with a large online crowdsourcing platform; thus, we expect it to prove useful to the wider computational linguistics research community and intend to release the code open source in the near future, likely as a repository on the U.S. Army Research Laboratory's GitHub site [6].

5. Future Work

We built CROWDTREE to explore the feasibility of crowdsourcing dependency parse annotation and to demonstrate the value of utilizing human-in-the-loop machine learning during the parse annotation process. Accordingly, we will be posting HITs on Mechanical Turk with a variety of parameterizations and publishing our findings. We hope to show that annotators without extensive linguistic training (e.g., Turkers) can quickly, cheaply, and accurately construct dependency trees and that integrated machine learning greatly benefits the overall process.

Furthermore, we intend to expand the system's machine learning capabilities by adding support for one or more active learning schemes. One obvious option would be to host multiple parsing models on the server and to select sentences for annotation based upon the level of disagreement between the parsers. We are considering the possibility of training one (or perhaps more) models per annotator during the annotation process, although implementing this in such a way as to avoid overtaxing available memory and computational resources may be a challenge.

A number of other extensions could be made to improve the tool's versatility in order to support a greater range of treebanking projects. We envision supporting empty nodes in order to handle traces and other phenomena, and we would like to support multiple syntactic word nodes for a given surface token, which would be valuable functionality for languages that make frequent use of clitics, such as Arabic. Finally, we hope to eventually expand the capabilities of CROWDTREE to support use cases beyond dependency parsing, such as part-of-speech tagging, semantic role labeling, or even language learning.

Acknowledgements

We wish to thank the anonymous reviewers for their comments and the Army Educational Outreach Program (AEOP) for the College Qualified Leaders (CQL) undergraduate internship program.

References

Böhmová, A., Hajič, J., Hajičová, E., and Hladká, B. (2003). The Prague Dependency Treebank. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, pages 103–127. Springer Netherlands.

de Castilho, R. E., Mujdricza-Maydt, E., Yimam, S. M., Hartmann, S., Gurevych, I., Frank, A., and Biemann, C.

[5]https://zombilingo.org/

[6]https://github.com/usarmyresearchlab/

(2016). A Web-Based Tool for the Integrated Annotation of Semantic and Syntactic Structures. In *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*, pages 76–84.

Fort, K., Guillaume, B., and Chastant, H. (2014). Creating Zombilingo, a Game With A Purpose for dependency syntax annotation. In *Proceedings of the First International Workshop on Gamification for Information Retrieval*, pages 2–6.

Fort, K., Guillaume, B., and Lefebvre, N. (2017). Who wants to play Zombie? A survey of the players on ZOMBILINGO. In *Proceedings of Games4NLP: Using Games and Gamification for Natural Language Processing*.

Gerdes, K. (2013). Collaborative dependency annotation. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 88–97.

Goldberg, Y. and Elhadad, M. (2010). An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In *Human Language Technologies: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2010)*.

Guillaume, B., Fort, K., and Lefebvre, N. (2016). Crowdsourcing Complex Language Resources: Playing to Annotate Dependency Syntax. In *Proceedings of the 26th International Conference on Computational Linguistics (COLING)*, pages 3041–3052.

Hajič, J., Smrz, O., Zemánek, P., Šnaidauf, J., and Beška, E. (2004). Prague Arabic Dependency Treebank: Development in Data and Tools. In *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*, pages 110–117.

He, L., Michael, J., Lewis, M., and Zettlemoyer, L. (2016). Human-in-the-Loop Parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP 2016)*, pages 2337–2342.

Laws, F., Scheible, C., and Schütze, H. (2011). Active Learning with Amazon Mechanical Turk. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011)*, pages 1546–1556.

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)*.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):330.

McDonald, R. and Nivre, J. (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 2007)*, pages 122–131.

Nivre, J. (2009). Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP 2009)*, pages 351–359.

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 (COLING 2008)*, pages 673–680.

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 2012)*.

Tratz, S. and Hovy, E. (2011). A Fast, Accurate, Non-Projective, Semantically-Enriched Parser. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1257–1268.

Tyers, F. M., Sheyanova, M., and Washington, J. (2018). UD Annotatrix: An Annotation Tool for Universal Dependencies. In *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories (TLT16)*, pages 10–17.

Von Ahn, L. (2006). Games with a Purpose. *Computer*, 39(6):92–94.

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 (ACL 2013): System Demonstrations*, pages 1–6.

Yimam, S. M., Biemann, C., de Castilho, R. E., and Gurevych, I. (2014). Automatic Annotation Suggestions and Custom Annotation Layers in WebAnno. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014): System Demonstrations*, pages 91–96.