Turk J Elec Eng & Comp Sci
() : –

A collaborative grammatical annotation tool for agglutinative languages

Received: .202 • **Accepted/Published Online:** .202 • **Final Version:** ..202

Abstract: The significance of high-quality treebanks has been steadily increasing due to their crucial role in developing natural language processing tools. The creation of such treebanks is an enormously labor-intensive and time-consuming task due to the need for qualified annotators and the quantity of data to be annotated. Tools that facilitate the annotation process have been developed, however, their interfaces are usually designed for analytic languages. Agglutinative languages tend to be richer in their morphology and require extensive annotations that significantly differ from annotations in analytic languages. Most of these tools are not well-suited for annotations in agglutinative languages such as Turkish. As such, low-resource agglutinative languages need suitable tools to accelerate the creation of high-quality data resources. This paper describes the design and implementation of a web-based collaborative grammatical annotation tool designed to expedite and facilitate the annotation of large treebanks, especially in agglutinative languages. This work builds on extensive experience with another grammatical annotation tool developed previously to annotate a Turkish treebank. The main objectives of this tool are to: (1) support the creation of valid and consistent annotations, (2) increase the speed of the annotation process, (3) improve the user experience of annotators, (4) support collaboration among annotators, and (5) provide an easily deployable and open-source web-based annotation tool with an application programming interface (API) to benefit the scientific community. This paper discusses the requirements elicitation, design, implementation, and evaluation of this tool along with examples.

Key words: Natural language processing, Linguistic annotation, Annotation tool, Web application, Dependency parsing, Universal dependencies

1. Introduction

Treebanks are significant resources in the development of natural language processing (NLP) tools and resources, and quality NLP resources need quality treebanks, manually annotated by linguistic experts. This is particularly true for agglutinative languages due to their complex morphologies. The creation of quality treebanks in morphologically rich languages (MRL) is highly labor-intensive and time-consuming due to the sheer amount of meticulous attention and effort required. Thus, annotation tools that support this process are essential. In recent years, there have been major efforts in increasing the data resources available for low-resource agglutinative languages –such as introducing new datasets and re-annotating existing treebanks. All these efforts require grammatical annotation by native speakers and/or linguistic experts.

Grammatical annotation involves marking each token of a sentence with corresponding linguistic values. Universal Dependencies [24] (UD) is one of the most widely accepted frameworks for morphosyntactic grammar annotations as it offers the flexibility and coherency needed to cover typologically very different languages in a unified way. The annotation format of UD called CoNLL-U [5] (Computational Natural Language Learning) defines a set of linguistic tags to express syntactic category (e.g. noun, verb, adjective) and morphological features of each token as well as the relationship between them. The CoNLL-U format has 10 columns for

each token of a sentence where the annotator can mark down word form, lemma form, syntactic category, dependency relation, and more, following predetermined rules and tags. For example, the universal part of speech (UPOS) column is used to denote the syntactic category of a token while the features (FEATS) column allows marking morphosyntactic features such as case, number, or tense. Since agglutinative languages allow stacking multiple morphemes on roots and stems for inflection, the FEATS column is heavily used in their annotations to mark several values for each token. This makes the effort required to annotate agglutinative languages significantly higher than analytic languages. Annotation tools with drag-drop and mouse-based interfaces, although appealing, are not well suited for agglutinative languages as they require alternating among input modalities frequently, disrupting the flow and slowing down the annotator.

Aware of this issue, BOAT-v1 [1] was developed to support the annotation of morphologically rich languages to produce treebanks compliant with the UD framework [24]. BOAT-v1 allowed annotations via using only the keyboard, yet the user experience revealed several points of improvement. The main takeaway from the feedback provided by the annotators that had used BOAT-v1 was a much better understanding of the time, effort, cognitive load, and extra information requirements of the annotation process. Improvements regarding these aspects should improve the performance of annotators, and consequently, produce higher-quality NLP resources.

This work presents a web-based collaborative grammatical annotation tool (BOAT-v2) that has been designed based on earlier experiences and requirement elicitation sessions with experienced annotators [1]. In light of the feedback from this experience, several user experience improvements were implemented to enhance the flow and consistency of the annotators. Additionally, this tool was designed as a web application that supports multiple users to enable a collaborative environment for annotators. The design and implementation of the tool aimed to: (1) support the creation of valid and consistent annotations, (2) increase the speed of the annotation process, (3) improve the user experience of annotators, (4) support collaboration among annotators, and (5) provide an open-source and easily deployable web-based annotation tool with an API to benefit the scientific community. The development started with requirements elicitation, for which earlier experiences and in-depth interviews with an experienced annotator were taken into account. The main issues identified during the requirement analysis phase are:

- The interface must be free of clutter and intuitive so that the annotator can focus on the task better.
- New features like auto-completion should be introduced to increase the speed of the annotator.
- Problems detected after extensive use of BOAT-v1 must be addressed.
- Support for collaboration has become significant since the increase in the effort to annotate large treebanks has led to the need for multiple annotators.

Considering the key points above, BOAT-v2 was designed in line with the typological particularities of agglutinative languages and the needs of annotators with experience in grammatical annotation. The resultant tool has been evaluated through several test rounds participated by both experienced and novice annotators. The aim of the test rounds was to assess how well BOAT-v2 was able to answer the requirements detected prior to the design and implementation phases, and whether newly introduced features and design choices improved the user experience. The feedback provided by the participants following the test rounds was overwhelmingly positive, which indicates that the ability to collaborate within the tool is essential to increasing the efficiency of multi-annotator treebank creation. The tool's final version has been made available [1] as an open-source resource.

[1] An earlier version of this paper was presented at the The International Conference on Agglutinative Language Technologies as a challenge of Natural Language Processing (ALTNLP) [1] in 2022.

The main contributions of this work are:

- Design and implementation of a grammatical annotation tool that supports the intensive cognitive effort expended by linguistic annotators based on requirements elicited from experienced annotators,
- Development of a web-based annotation tool based on a supporting API for programmatic access and extensibility,
- Multi-user support to provide individual spaces for annotations, computation of inter-annotator agreements, and other potential collaboration, and
- Release of an accessible tool that is an open-source resource, virtualized via Docker [13] for easy deployment.

The remainder of this paper is organized as follows: Section 2 lays the background of the proposed tool, Section 3 presents related work, Section 4 describes the requirements and design, Section 5 presents the implementation and features, Section 6 illustrates a use case of annotation, Section 7 explains 4 different usability evaluations performed on the tool and Section 8 provides a discussion along with our conclusions.

2. Background

2.1. BoAT-v1

BoAT-v1 [1] is a standalone tool for annotating treebanks compatible with the UD framework [24]. Although it was specifically developed for annotating Turkish treebanks, it supports agglutinative languages with rich morphologies; thus, it can be used for other languages as well. It allows annotating one treebank at a time. Annotations are stored in a file in the CoNLL-U format. The file is updated during the annotation process. It uses a validation script developed by UD to dynamically display errors to the annotator as they annotate. BoAT-v1 was used to create a manually annotated Turkish dependency treebank comprising 9,761 sentences from 5 different domains: essays, national newspapers, instructional texts, popular culture articles, and biographical texts. It was implemented using Python [20] and Qt [21].

For annotation of any sentence, the annotator is shown a table which has a token per row with its corresponding columns (ID, FORM, LEMMA, UPOS, XPOS, FEATS, HEAD, DEPREL, DEPS, and MISC as detailed in [1]). The annotator manually enters values for each column of each token. It supports splitting and joining of lemmas. This is particularly significant for agglutinative languages since their tokens are often comprised of multiple morphemes whose accurate annotations sometimes require adjusting lemma boundaries by splitting or merging. Tokens that are split result in additional two rows whose lemmas are then manually adjusted by an annotator such that they become distinct parts of a token. Figure 1 shows an annotation of a sentence. The token "yoktu" (ID: 4-5) is split into "yok" (ID: 4) and "tu" (ID: 5). Furthermore, with this split, the annotator divides the FEATS column's value into individual morphological features for the lemmas with IDs 4 and 5.

BoAT-v1 supports viewing features and their values individually under their corresponding columns (e.g. Case=Nom|Number=Sing|Person=3 can be shown in columns "Case", "Number", and "Person" with the values "Nom", "Sing", and "3"). It also allows the annotators to be able to take notes for specific annotations.

2.2. Challenges of Agglutinative Languages

Agglutinative languages allow stacking multiple morphemes on a single root or stem to convey a wide range of linguistic information such as tense, aspect, number, person, possessor, and more. Analytic languages, on the other hand, convey the same information through often free-standing function words. A clear illustration of this

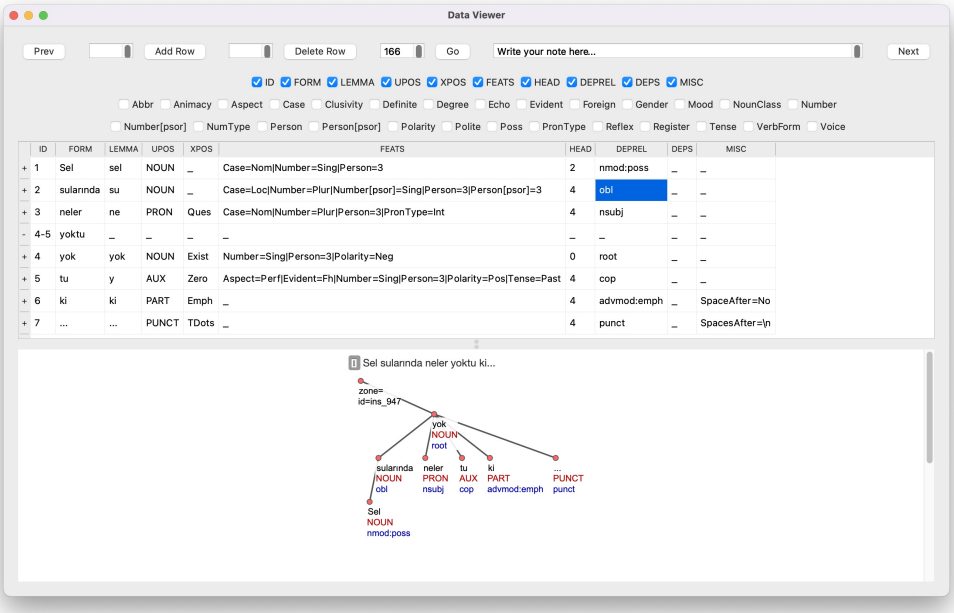


Figure 1. The annotation screen on BOAT-v1, captured while the sentence "Sel sularında neler yoktu ki..." is being annotated.

contrast can be observed in Sentence (1): Each lemma of the Turkish sentence stacks at least two inflectional or derivational morphemes that carry out the function of free-standing morphemes in the corresponding English sentence. Hence, the Turkish sentence has only 3 forms *(masadaki, kitaplarını, görmüştüm)* while its English translation has 10 forms *(I, had, seen, your, books, that, were, on, the, table)*.

(1) Masa-da-ki kitap-lar-ın-ı gör-müş-tü-m.
table-LOC-DER book-PL-POSS.2SG-ACC see-ANT-PST-1SG[2]
'I had seen your books that were on the table.'

As a result of this contrast, lemmas in agglutinative languages like Turkish bear more features in the FEATS column compared to analytic languages like English. In addition to a higher feature-per-token count, agglutinative languages often have higher non-unique feature counts as well due to their rich repository of derivational and inflectional morphemes.

In order to offer an empirical example of this contrast, we compared non-unique features, total tokens, and features-per-token counts of multilingual UD treebanks (ATIS and PUD) [27–30]. Despite following different annotation customs, both Turkish treebanks have significantly higher non-unique features and lower token counts than their English counterparts (see Table 1). As a result, both Turkish treebanks have more features per token; in fact, ATIS Turkish treebank has more than two times more features per token than ATIS English.

In addition to having a higher feature count, being a free word order language makes annotating Turkish treebanks substantially challenging. As Turkish allows scrambling and topicalization, the average dependency length per token in Turkish sentences is much higher than in fixed word order languages like English. Moreover,

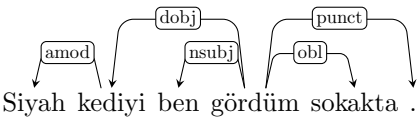
[2] 1 = first person, 2 = second person, ACC = accusative, ANT = Anterior, DER = Derivation, LOC = locative, PL = plural, POSS = possessive, PST = past, SG = singular.

Comparison of ud Treebanks			
Treebank	**Non-unique feats**	**Total tokens**	**Feats per token**
ATIS Turkish	112,214	45,875	2.45
ATIS English	63,434	61,879	1.03
PUD Turkish	32,583	16,536	1.97
PUD English	23,995	21,176	1.13

Table 1. Comparison of morphological feature annotations of Turkish and English UD treebanks with equivalent sets of sentences in terms of meaning.

such word order changing operations also affect headedness and the dependency arc direction while changing dependency lengths: The order of a dependent and its head can vary within or across the sentences (compare Sentence 1 and Sentence 2).

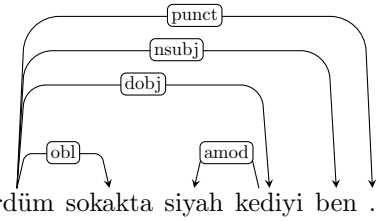
(2)



Lit. The black cat I saw on the street.
'I saw the black cat on the street.'

Average arc length: 1.4
Longest arc length: 3

(3)



Lit. Saw on the street the black cat I.
'I saw the black cat on the street.'

Average arc length: 2.8
Longest arc length: 5

An annotation interface designed for agglutinative languages must facilitate the annotator's job despite the changes in dependency length and arc direction. As the dependency length increases, drag-drop interfaces become impractical since they require the annotator to drag lemmas for longer distances, thus decreasing the annotation speed and increasing the effort. Moreover, the need to annotate more features per token also makes drag-drop interfaces very difficult to use for annotating agglutinative languages. Thus, opting for an interface that utilizes the keyboard and allows typing is more practical for such languages.

In addition to the data entry method (mouse versus keyboard), feature-per-token counts, and changes in the dependency length and arc direction also require some adjustments in the interface. For instance, the annotation interface must display the entire sentence and all lemma ids at all times. This way, the annotator can see the head and/or dependent of the lemma that is being annotated even though they are dealing with a very verbose sentence or a very high dependency length. Moreover, the annotation interface must be able to display large numbers of tags in the features column in an easy-to-read way so that the annotator can keep track of the annotation even when they are dealing with particularly packed lemmas.

3. Related Work

Annotation tools can be categorized in terms of their accessibility, and the support they provide for different languages, various annotation frameworks, user interface modalities, annotation-related standards, multiple annotators, searching of annotations, and automation. Adherence to standards is recommended to get the most benefit from the annotated language resources. Universal Dependencies [24] (UD) is an actively growing

standard for the annotation of treebanks, intending to cover all languages, and its support for agglutinative languages is evolving.

Numerous dependency annotation tools have been developed with many of them being UD-compliant, such as brat [2], UD Annotatrix [7], and DGA [10]. Most of these tools are capable of annotating a variety of languages and all the tools that are detailed below do so. Most tools require mouse-based user interactions, which is inefficient for the annotation of agglutinative languages due to several reasons, including the need for extensive annotation for most tokens.

ITU Treebank Annotation Tool [4] is an open-source standalone tool, developed specifically for Turkish. It has three stages of annotation: morphological analysis, morphological disambiguation, and syntax analysis. It offers semi-automated support for annotators through analyzers for creating new datasets as well as correcting already existing Turkish treebanks. It mostly relies on mouse-based interactions and doesn't have support for the UD framework. This tool has been used to annotate the ITU Web Treebank [4]. It has three versions, and its latest version (3) was written in Java, requiring Java to be installed on the system used for annotations.

brat [2] is a web-based *text* annotation tool which provides a collaborative annotation environment. Being web-based, it's usable across different platforms. It's very flexible in its annotation types and has support for many annotation tasks such as dependency, entity, or coreference annotation. This tool is also used for the visualization of annotations. Annotators use a mouse and keyboard together to annotate. Annotators are able to transfer their annotations in a file format called *standoff* specific to brat. It is configurable to support annotations compliant with UD and is used for visualization in UD's own documentation system. brat supports real-time collaboration, even allowing annotators to edit the same annotation. It supports searching the documents or their annotations through using various filters.

The Dependency Grammar Annotator [10] (also known as DGA or DgAnnotator) is a dependency annotation tool. It enables its users to annotate POS and dependency relation tags via mouse clicks. It supports the transfer of annotations in UD-compliant formats. It doesn't support multiple annotators to work on the same treebank. There is no search functionality. It was written in Java [19] and consequently requires Java.

WebAnno [26] is a web-based open-source annotation tool, that is not restricted to dependency annotations but has support for morphological, syntactical, and semantic annotations also, with multi-user support. To annotate the morphological features of a token, it requires several mouse clicks which is impractical for MRLs. The sentences to be annotated are displayed vertically on a single screen, unlike many tools that focus on a single sentence at a time. It supports the curation of annotations created by multiple annotators. It has two installment options, standalone and server. The standalone version requires Java, while the server version requires a Tomcat [18] server with a MySQL [17] database.

UD-Annotatrix [7] is a UD-specific dependency annotation tool that aims to be simpler than brat. Mouse usage is necessary, in addition to keyboard entries, for annotation. It supports real-time collaboration if the tool is deployed in a server. It employs annotation standards of UD and CG3 [9]. It doesn't have a search functionality.

CoNLL-U Editor [3] is a web-based annotation tool, specific to the CoNLL-U [5] format of UD. It saves every edit of annotations by version control. If specific lists of part-of-speech tags are provided by the annotators, it supports auto-completion during annotation. Multiple annotators are able to annotate the same treebank if they don't work on the same sentence at a time. Annotators can search using any combination of linguistic tags of UD using the tool. It requires a Java-based server.

StarDust [6] is a recent standalone dependency annotation tool. It uses 2 layers: morphological disambiguation and dependency annotation layers, sequentially. In the dependency annotation layer, annotators are expected to click on a dependent and drag the cursor to its head. It stores each sentence's annotation in a different file. For backups of the annotations, it uses Dropbox [15], which is a proprietary cloud storage platform. There were several treebanks annotated using this tool: 1 English and 5 Turkish treebanks.

BoAT-v2 is the second iteration of BoAT-v1. It improves on it by including a search functionality, an auto-completion feature, a database to represent sentences in a more essential way rather than a plaintext file, multiple dependency graph visualization options, and a web interface with an API for flexibility and accessibility. BoAT-v2 also reduces the clutter that BoAT-v1 was reported to have in parts of the screen in the feedback of the previous BoAT-v1 usage.

Comparison of Annotation Tools					
Tool	**Mouse**	**Keyboard**	**Standards**	**Multi-user**	**Search**
brat	+	+	Extendable	+	+
UD Annotatrix	+	+	UD & CG3	+	-
DgAnnotator	+	-	UD	-	-
CoNLL-U Editor	+	+	UD	+	+
ITU	+	+	UD	-	-
WebAnno	+	+	UD	+	-
StarDust	+	+	-	-	-
BoAT-v2	+	+	UD	+	+

Table 2. Properties of the above-mentioned annotation tools, namely whether they support mouse or keyboard usage, what standards they abide by, and whether they support multi-annotator treebanks or a search functionality.

4. Requirements and Design

Requirements elicitation and validation meetings were held with annotators who worked on a treebank [1] involving thousands of sentences to identify the software requirements. The main requirements that emerged are:

Support for sentence annotation: To ensure speedy and accurate annotations, various features must be provided. In order to eliminate mistakes caused by typos while saving time for the annotator, *auto-completion* should be supported whenever possible. Due to typologies of and syncretism being generally higher in MRLs, automatic parsing of such treebanks is difficult and they more often than not fail to create valid parsings of tokens. Thus, refining automatic parsing becomes an important task for agglutinative languages. Words in MRLs tend to have morphemes stacked on roots. Annotators must be able to refine or correct automatically parsed entries, which for agglutinative languages includes splitting or joining of lemmas.

Use of screen real estate and customization: Annotators annotate many sentences in each session. For agglutinative languages, sentences tend to be long and complicated. The view of such sentences can cover a considerable part of the screen. The annotator must focus on the sentence being annotated, the lemmas, and all the features, which require lots of concentration. The annotation process must require little to no scrolling. For particularly long sentences, no scrolling may not be possible; however, the sentence being annotated should always be in sight of the annotator. Furthermore, every annotator has their unique preferences regarding how they would like to annotate. They must have some control on showing or hiding certain information according to their preferences. Overall, efficient use of screen real estate is important to provide the context needed to annotate long sentences.

Keyboard-oriented input: Annotations of agglutinative languages require annotation of several morphological features for many tokens of a sentence in comparison to analytic languages like English. While drag-drop interfaces can work well for analytic languages, they don't for MRLs as they require more toggles between mouse and keyboard interactions which becomes frustrating. Keyboard-based interaction must be offered for all possible tasks so that the annotator can work without needing to switch between the keyboard and the mouse.

Collaborative annotation: The large sizes of treebanks necessitate that they be annotated by multiple annotators. The tool must support multiple annotators working simultaneously on the same treebank. The ability to share annotations is vital for reference and consistency. Hence, the tool must provide a means for registering multiple annotators and keeping track of their respective annotations.

Search: During the annotation of a sentence, annotators sometimes need to refer to previous annotations for guidance. An annotator should be able to search a treebank according to surface forms, and importantly, according to linguistic features. They should be able to perform complex searches that involve several criteria such as UD tags, individual morphological features, and surface forms. This requirement reduces cognitive load and facilitates consistency among annotations.

Given these requirements, we decided that a web-based application that supports multiple authenticated users would be able to provide a collaborative platform for annotators. The treebank and user annotations persists in a database which makes managing the data and searches much more reliable and efficient. Also, an API is developed to support flexibility and extensibility. Finally, in addition to making the tool open-source, we containerized the application using Docker [13] to support its accessibility.

5. Implementation

The annotation tool is implemented using Python [20], the web application development framework *Django* [11] and the API framework *Django REST Framework* (DRF) [14]. The webpages use Bootstrap [8]. PostgreSQL [16] is used as the database management system. The database models are in line with the UD format of sentences. Annotations are validated according to the UD guidelines and errors are reported on the annotation page. Three alternative forms of dependency graph visualizations are supported, two of which are newly added, compact and horizontal to reduce the required screen real estate [22, 23].

The following features have been implemented in order to support the creation of valid annotations with increased speed in a collaborative manner:

Treebank handling: The tool should support the annotation of multiple treebanks. BOAT-v2 uses a database to persist the annotations of multiple annotators of multiple treebanks. After annotations are done, annotators can export their annotations as an annotation file in the CONLL-U format of UD.

Sentence annotation: A sentence is selected by a user in order to annotate. The annotation page consists of three main parts: (1) A table with rows for each token of a sentence and columns, which represent the UD tags of a sentence, corresponding to their annotations; (2) the dependency graph of the sentence; and (3) validation results according to the UD guidelines. The dependency graph and errors are updated with the annotations. Several dependency graph representations are supported to serve the annotator's preference. Two horizontal dependency graphs are supported since vertical graphs can consume a significant amount of screen real estate, which leads to loss of focus in long sentences commonly found in agglutinative languages. One vertical tree-like dependency graph is also supported in case the annotator would like to see the tree structure of the annotation.

An annotator may need to stop the annotation of a sentence due to sentence complexity or external interruptions. To capture the state of an annotation, status is used with values of "New", "Draft" and "Complete". The status of a sentence that has not yet been annotated is "New". Annotators set their annotation statuses to "Draft" or "Complete".

The annotator is able to perform most operations via keyboard action, based on the demand of the annotators (see Section 4 for details). Upon the experience of annotating a Turkish treebank with BoAT-v1, the annotators were pleased with the convenience and speed keyboard-based interaction provided them with.

Searching for reference and consistency: To facilitate effective searching, a search functionality has been implemented in BoAT-v2. Users are able to search for previously annotated sentences by any annotator based on combinations of surface text and UD tags (e.g. morphological features) with regular expression support. This feature supports annotators to share their annotations, which consequently is expected to result in more valid and consistent treebanks. Without a search feature, an annotator would have to manually search their annotation files for relevant cases (e.g. how to annotate a certain surface form's UPOS tag). The form might have been annotated in various ways throughout the treebank, alas it is highly unlikely that a manual search would reveal this inconsistency. In this case, the annotator would likely use the first usage they come across as their reference. This problem gets more complicated for syncretic morphemes such as *-ki* in Turkish. For instance, in the sentence "Evdeki halılar yıkandı." (*The rugs at home were washed.*), the *-ki* acts as an adjectivizer. However, in "Benim halılarım yün, Ayşeninkiler sentetik." (*My rugs are woolen. Ayşe's are synthetic.*), it becomes pronominal. Searching for sentences where the morpheme *-ki* occurs with basic text search of an annotation file would be unfruitful as there would be too many occurrences, owing to its role in the Turkish language's grammar and also brevity as a word.

Inter-annotator agreement: The annotator consistency of a treebank is an indicator of the quality of the resulting resource. Inter-annotator agreement is done by computing the consistency among annotators. Since this tool keeps track of annotator actions, such computations are straightforward. Some visualizations shall accompany these statistics.

6. Using BoAT-v2

A typical annotation can proceed as follows: An annotator selects a sentence from a treebank. An annotation table appears with the sentence parsed according to the UD format. Each row corresponds to a token and its annotations. Figure 2 shows the annotation view while an annotator is annotating a Turkish sentence "Sel sularında neler yoktu ki..." (translation: *What wasn't in the flood waters...*).

An annotator can make use of dependency graphs, errors, and search during annotation. Dependency graphs are visual cues for how lemmas are dependent upon one another. Errors are helpful reminders compliant with UD. Searching can help the annotator disambiguate ambiguous forms, or check previously annotated sentences to ensure coherency and inter-annotator agreement. The annotator can search for previously made annotations in combinations of text and feature values.

Annotators can customize the columns and dependency graph in accordance with their preferences. They can hide and view columns as they prefer, and they can choose from several different dependency graph styles. When an annotation is finished, its status can be set to "Complete".

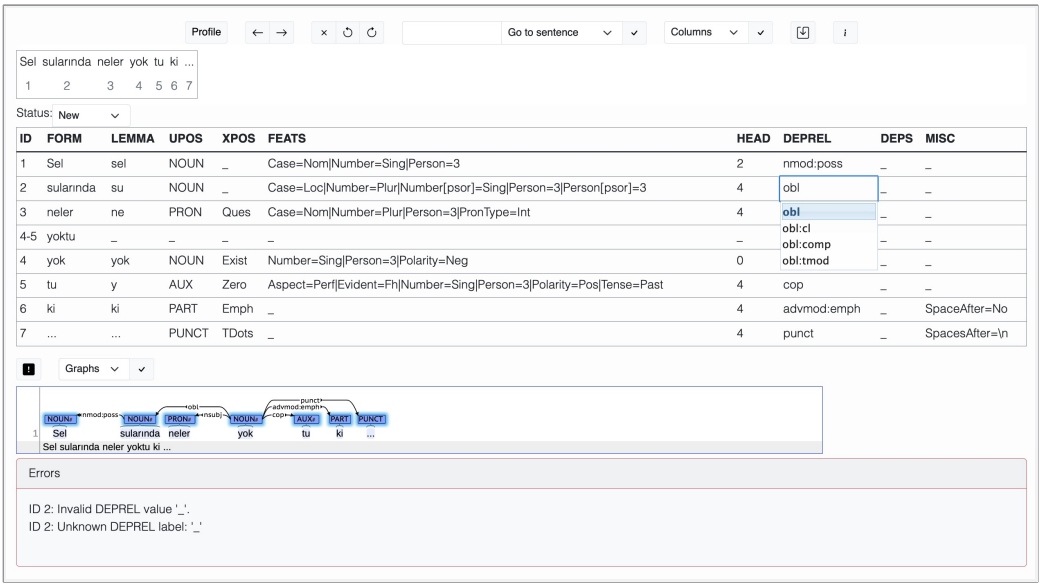


Figure 2. The annotation screen captured while the sentence "Sel sularında neler yoktu ki..." *('What wasn't in the flood waters...')* is being annotated. The DEPREL tag for the surface form "sularında" is being annotated by selecting among the valid alternatives that appear in the pop-up. Selections can be made with the use of arrow keys.

7. Evaluation

Following the development phases of BoAT-v2, 4 test were conducted in order to evaluate the tool. The aim of each test was to assess a different aspect of BoAT-v2.

7.1. Version Comparison

The first test was carried out in June 2022 to compare BoAT-v1 and BoAT-v2. As part of this test, two sets of 10 sentences were created. Each set contained unique sentences with each sentence in the first set having another sentence with the same amount of tokens in the second set. The reason behind this choice was to eliminate potential confounders so that the two versions of the tool could be compared in terms of annotator speed, responsiveness, and various other metrics.

A single annotator used BoAT-v1 to annotate the first set, then used BoAT-v2to annotate the second set. During both of the annotation processes, an observer took notes and timed how long it took to complete each sentence on each tool. There was a noticeable speedup (approximately 30%) using BoAT-v2. Among the new features that were most appreciated are auto-completion, condensed dependency tree representation, a significant reduction in scrolling, keyword search, and searching by morphological features. The non-search-related features proved to be instrumental in retaining focus.

7.2. User Testing

After version comparison, several test cases were carefully designed for user testing in July 2022. These cases included testing various functionalities of the tool, such as creating annotator accounts, creating and uploading treebanks, annotating both simple[3] and complex sentences[4], splitting and merging lemmas, changing preexisting

[3]Simple sentence annotation refers to sentences with less than 10 tokens.
[4]Complex sentence annotation included long sentences which also required splitting lemmas.

annotation tags, viewing different dependency graphs, and searching in previously created annotations.

The goal of the test was to fully assess the responsiveness and functionality of BOAT-v2. Thus, the cases covered both newly introduced functions like choosing different dependency graph styles and old functions like lemma splitting. Three experienced linguist annotators carried out the user tests on different operating systems and browsers (Safari [32] on macOS [31], Chrome [34] on Windows [33]), all on desktop devices. Following the tests, they filled out questionnaires to give feedback on whether they were able to carry out the annotations and operations they were expected to. In addition, they reported any errors or issues they came across during the testing. The test results were mostly matching what the expected results were. The feedback provided by the participants was referred to while improving the tool further and making it more robust.

7.3. Automated Testing

After user testing, we wanted to construct automated testing scripts in order that the development does not produce unintended results, such as preventing a feature from working expectedly. For this, we have designed and tested with several test cases via Selenium [35] in September 2022. In these tests, we aimed to test if the tool's interface works as expected in various use cases, leveraging assertions in the tests. These use cases included logging in, creating and deleting treebanks, and searching sentences. The actual results of these tests were matching the expected results. We plan to use these automated testing scripts when we update the corresponding web pages in these tests so that the development continues to improve the tool.

7.4. Learnability and Intuitiveness

The final round of testing targeted the learnability and intuitiveness of BOAT-v2. For this purpose, a test case similar to the previous user testing was designed and carried out in December 2022. This test case included numerous functionalities of the tool, such as creating and logging into a user account, creating and uploading a treebank, annotating parts of simple and complex sentences, changing preexisting annotations, merging lemmas, and searching in current and previously annotated treebanks.

Two linguists [1] with no annotation experience participated in the test. Before they started the test, they were introduced to the tool and the UD framework by an experienced annotator for the first time. They were shown how to use the tool and do annotations, and then they were given the dataset to be annotated. They did the tests on the same device and browser they received the training on (Safari on a desktop macOS device).

As they carried out the tests, the participants were observed by the trainer who also took notes regarding the tests. The participants were able to ask questions and get directions from the trainer regarding both the UD framework and the annotation tool. After testing, the participants were asked to fill out a questionnaire regarding their experience. Their feedback and notes taken by the expert during the tests were considered together by the development team.

8. Discussion and Conclusions

Our aim with BOAT-v2 is to develop a collaborative web-based application based on previous experiences, supporting the annotation process of many languages, specifically morphologically rich ones. We believe that having experts in linguistics and annotators experienced in agglutinative treebank creation was instrumental in understanding the requirements and also in the design process. We held numerous elicitation interviews and further meetings for clarifications.

We have used modern software development tools and management practices during the development lifecycle of this tool. The API enables various extensions of the tool and also programmatic access to the treebanks. The containerization with Docker [13] eases the deployment procedure. It is available [1] as an open-source resource. The wide availability of the tool is valuable for future use and developments.

We are encouraged by the evaluation performed on this tool and anticipate its extensions. The feedback coming from the evaluation presents many ways that the tool can be improved. For the tool's future, we plan on improving the current functionality, implementing annotation-centric new features, such as automated annotation of certain tags based on rules provided by annotators, or creating notes on annotations to communicate with other annotators through the application. We also plan to implement support for various languages regarding auto-completion and annotation validation.

References

[1] Anonymous. We're not including this reference to preserve anonymity for the review process.

[2] Stenetorp P, Pyysalo S, Topić G, Ohta T, Ananiadou S et al. brat: a web-based tool for nlp-assisted text annotation. In: Proceedings of the Demonstrations Session at EACL; Avignon, France; 2012. Association for Computational Linguistics.

[3] Heinecke J. Conllueditor: a fully graphical editor for universal dependencies treebank files. In: Universal Dependencies Workshop; Paris; 2019.

[4] Pamay T, Sulubacak U, Torunoğlu-Selamet D, Eryiğit G. The annotation process of the itu web treebank. In: Proceedings of The 9th Linguistic Annotation Workshop; Denver, Colorado, USA; 2015. Association for Computational Linguistics, pp. 95-101. https://doi.org/10.3115/v1/W15-1610

[5] Buchholz S, Marsi E. Conll-x shared task on multilingual dependency parsing. In: Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X); New York City; 2006. Association for Computational Linguistics, pp. 149-164.

[6] Yenice AB, Cesur N, Kuzgun A, Yıldız OT. Introducing stardust: a ud-based dependency annotation tool. In: Proceedings of the 16th Linguistic Annotation Workshop (LAW-XVI) within LREC2022; Marseille, France; 2022. European Language Resources Association; pp. 79-84.

[7] Francis MT, Mariya S, Jonathan NW. Ud annotatrix: an annotation tool for universal dependencies. In: Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories (TLT16); 2018; pp. 10-17.

[8] Bootstrap (2022). Bootstrap [online]. Website https://getbootstrap.com/docs/5.1 [accessed 05 May 2022].

[9] Didriksen T (2022). Constraint Grammar Manual [online]. Website https://visl.sdu.dk/cg3/single/ [accessed 18 November 2022].

[10] Attardi G (2022). DgAnnotator [online]. Website http://medialab.di.unipi.it/Project/QA/Parser/DgAnnotator/ [accessed 27 March 2022].

[11] Django Software Foundation (2022). Django documentation [online]. Website https://docs.djangoproject.com/en/4.0 [accessed 25 March 2022].

[12] Alex Gaynor, Carlton Gibson and others (2022). django-filter [online]. Website https://docs.djangoproject.com/en/4.0 [accessed 24 April 2022].

[13] Docker Inc. (2022). Docker [online]. Website https://www.docker.com [accessed 2 May 2022].

[14] Encode OSS Ltd. (2022). Django REST framework [online]. Website https://www.django-rest-framework.org [accessed 27 March 2022].

[15] Dropbox, Inc. (2022). Dropbox.com [online]. Website http://dropbox.com [accessed 18 November 2022].

[16] The PostgreSQL Global Development Group (2022). PostgreSQL [online]. Website https://www.postgresql.org/docs [accessed 27 March 2022].

[17] Oracle (2022). MySQL [online]. Website https://www.mysql.com/ [accessed 16 December 2022].

[18] The Apache Software Foundation (2022). Apache Tomcat® [online]. Website https://tomcat.apache.org/ [accessed 16 December 2022].

[19] Oracle (2022). Oracle Java Technologies [online]. Website https://www.oracle.com/java/technologies/ [accessed 16 December 2022].

[20] Python Software Foundation (2022). Python [online]. Website https://www.python.org [accessed 24 April 2022].

[21] The Qt Company (2022). Qt [online]. Website https://www.qt.io/ [accessed 25 May 2022].

[22] Explosion (2022). spaCy [online]. Website https://spacy.io [accessed 27 March 2022].

[23] Pyysalo S (2022). conllu.js [online]. Website http://spyysalo.github.io/conllu.js [accessed 25 March 2022].

[24] Universal Dependencies (2022). UD [online]. Website https://universaldependencies.org [accessed 2 May 2022].

[25] Universal Dependencies (2022). UD tools [online]. Website https://universaldependencies.org/tools [accessed 22 April 2022].

[26] WebAnno (2022). WebAnno [online]. Website https://webanno.github.io/webanno/documentation [accessed 5 May 2022].

[27] Universal Dependencies (2021). UD English Atis [online]. Website https://universaldependencies.org/treebanks/en_atis/index.html [accessed 4 Jan 2023].

[28] Universal Dependencies (2021). UD Turkish Atis [online]. Website https://universaldependencies.org/treebanks/tr_atis/index.html [accessed 4 Jan 2023].

[29] Universal Dependencies (2021). UD English PUD [online]. Website https://universaldependencies.org/treebanks/en_pud/index.html [accessed 4 Jan 2023].

[30] Universal Dependencies (2021). UD Turkish PUD [online]. Website https://universaldependencies.org/treebanks/tr_pud/index.html [accessed 4 Jan 2023].

[31] Apple Inc. (2022). macOS [online]. Website https://www.apple.com/macos/ [accessed 5 Jan 2023].

[32] Apple Inc. (2022). Safari [online]. Website https://www.apple.com/safari/ [accessed 5 Jan 2023].

[33] Microsoft (2022). Windows [online]. Website https://www.microsoft.com/en-us/windows [accessed 5 Jan 2023].

[34] Google (2022). Google Chrome [online]. Website https://www.google.com/chrome/ [accessed 5 Jan 2023].

[35] Selenium (2022). Software Freedom Conservancy [online]. Website https://www.selenium.dev/ [accessed 5 Jan 2023].