

UDConcord: A Concordancer for Universal Dependencies Treebanks

Lucas Gabriel Mendes Miranda and Thiago Alexandre Salgueiro Pardo

Interinstitutional Center for Computational Linguistics (NILC)
Institute of Mathematical and Computer Sciences, University of São Paulo
São Carlos – SP, Brazil
lucasgmm@usp.br, taspardo@icmc.usp.br

Abstract. This paper presents UDConcord, a concordancer web application. The tool is designed to be visual and easy-to-use, working with treebanks that were annotated using the Universal Dependencies international model. It allows users to upload a treebank in the CoNLL-U format. After the upload, users can search for terms and linguistic categories of interest in the treebank. Because the tool is a concordancer, the search results are composed of sentences with occurrences of the searched elements displayed in a concordance list. That means that each sentence with a matching term will be displayed in a row, with the found term centralized and highlighted, accompanied with other information selected by the user to be visualized. UDConcord also allows users to easily modify sentence annotation. Finally, UDConcord makes it possible for users to download the treebank’s updated version, with every change made up until that point.

Keywords: Concordancer · Universal Dependencies · Treebank.

1 Introduction

The Universal Dependencies project (UD) [6, 5] is an initiative that seeks to standardize dependency-based treebank annotation. The project establishes a set of guidelines that must be followed during the construction of these types of treebanks.

One of the UD guidelines states that a treebank must be represented in CoNLL-U file format. Such files must contain all the sentences in the treebank, along with its annotation data, which is token/word-based. According to the guidelines, every word/token in a sentence must have a series of values distributed over 10 fields (columns), including the word’s part-of-speech tag, lemma, dependency relation, and other properties.

Those guidelines introduce a series of benefits to treebank annotation and design, but, to people who are manipulating the CoNLL-U files, they also can be hard to deal with. Particularly, problems might arise when users are trying to query a treebank. For example: if a user wanted to search for two consecutive words with specific part-of-speech tags in a treebank, he/she would not be able

to do it easily, thanks to the fact that treebanks are formatted in a table-like manner and that the available search tools are not straightforward to use or to install.

To help in this front, we present UDConcord, which is a concordancer web application to simplify search, analysis, and edition of CoNLL-U files. UDConcord seeks to be easy to use and simple (without the need to install the tool or to learn some kind of search syntax), allowing users to query treebanks, presenting the query results in a form a concordance. It also allows users to edit sentences' annotation data easily.

2 Related Work

There are a number of applications that allow users to query treebanks. Unfortunately, most of them have at least one of the following two issues:

- They do not allow users to edit the treebanks. For example, TüNDRA [4] and Grew-match [2] are both tools for *only* searching treebanks;
- They have user interfaces that are relatively complex, which makes it harder for inexperienced users to use them. For example, ConlluEditor [3] is one of the tools that offer both searching and editing of treebanks. However, its user interface suffers from an excess of buttons and options, which can be confusing for new users. Other interesting example to cite is Arborator-Grew [1], which, besides intended to be more user friendly and to include more functionalities (allowing to search, edit and visualize trees), requires the user to master some search syntax and to use an interface with too much information.

These two problems were considered when designing UDConcord, especially the second one. Our main goal was to create a tool that is simple, intuitive, and easy to use, in order to make it friendlier to inexperienced users.

Below, we highlight two tools that are similar to ours and that somehow inspired us to propose our tool, pointing UDConcord's strengths in comparison.

Interrogatório [8] is an environment for searching and editing universal dependency-based treebanks. It is written with Python and Javascript. However, the app is supposed to only run locally and to be accessible with a web browser through localhost. That could be a problem, because, in order for the user to use Interrogatório, he/she must install it first, which can be confusing if they are less experienced. UDConcord does not suffer from this, because it is a web application accessible through the web.

To make queries with Interrogatório, the user must learn a query language that is similar to Python code. Again, if the user is unfamiliar with Python, he/she might experience some confusion in the process of learning the language. To avoid this possibility during the usage of UDConcord, we implemented the query feature with the use of a form. That is, to make queries with UDConcord, users do not need to learn a query language, they only need to fill a form with their query's parameters and click on the search button.

Another tool very similar to UDConcord is Grew-match [2]. Like Interrogatório, it is written in Python and Javascript and allows users to query treebanks using a query language, which we already commented that can be confusing for less experienced users.

Like UDConcord, Grew-match is a web application accessible through the web. However, Grew-match’s users can only query a set of predefined UD-based treebanks, which is fairly large, while in UDConcord’s users have to upload their treebanks in order to query them. We believe that this is useful because it offers more flexibility to users.

Another important distinction between UDConcord and Grew-match is that the latter does not allow its users to edit the CoNLL-U from the queried treebank, while UDConcord does.

3 The Architecture of UDConcord

UDConcord is an application with three main components: a back-end, a front-end, and a reverse proxy. All these three components are installed in different Docker containers, like shown in Figure 1.

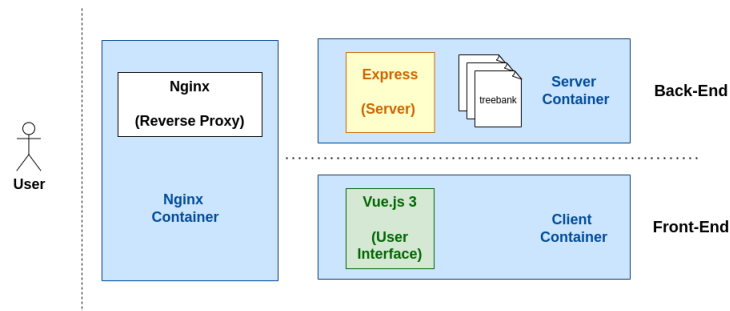


Fig. 1. UDConcord’s Architecture.

In the following subsections, we will specify what exactly each component does.

3.1 Back-end

The back-end contains code concerned with the search and storage of treebanks. It is entirely built with Javascript, with the *Node.js* runtime.

When the user uploads a treebank to our application, the back-end is responsible for parsing it to an array of Javascript objects (with each object being a sentence from the treebank) and saving it as a JSON file.

When the user searches a treebank, the back-end reads the JSON file that corresponds to the treebank and looks for sentences that satisfy the user’s search

conditions. Once they are found, they are sent back to the user. This approach guarantees queries with reasonable wait times for CoNLL-U files with at least 20 MB of size.

All these functionalities are exposed to the front-end through an API built with the *Node.js* framework *Express*.

3.2 Front-end

The front-end is composed of code concerned with the user interface. More specifically, it contains all the HTML, CSS and Javascript used to build it. It is important to note, that, for the Javascript language, we used the *Vue.js 3* framework, which immensely facilitated our work. Furthermore, certain components, like text inputs, tables, and buttons were taken from the *PrimeVue* component library. The front-end communicates to the back-end through HTTP requests to the API in the latter.

3.3 Reverse Proxy

The reverse proxy is a server responsible to direct users' requests to another appropriate server. For example, when the user enters UDConcord's web address, he will request the server holding the static files (HTML, CSS, and Javascript files) to download them. The reverse proxy will receive this user's request and direct it to the server running in the client container. Similarly, when the user makes a request to the back-end API, the reverse proxy will direct it to the Express server, which will process the request.

Our reverse proxy is configured using *NGINX*, which is a software commonly used for this purpose.

4 The Concordancer

UDConcord has four main features:

- The possibility of searching for terms and linguistic categories of interest in an uploaded treebank;
- To display every sentence with an occurrence of the searched elements;
- The possibility of editing the corresponding CoNLL-U files;
- Enabling the user to download the updated version of the treebank (with all the made changes);

All of these features are further described in the following subsections.

4.1 Searching/querying

UDConcord’s main feature is to let users query a treebank. However, to do that, they must first upload the treebank to the system. The upload screen is available on the home page of the app. It is the first thing users see when they enter the website.

To upload the treebank, users just have to click on the blue button labeled *Choose file* and select in their operating system the CoNLL-U file that represents the treebank.

Simple Searches After uploading the treebank, users will be redirected to the search screen. In that screen, he/she will have to specify their query parameters filling out a search input, which is shown in Figure 2.

Fig. 2. Search input.

The selector highlighted in red allows the user to choose between five properties (all defined in the UD specifications) to search for in the uploaded treebank: forms, lemmas, part-of-speech tags (POS tags), dependency relations (deprels) and features (feats).

In addition, the selector highlighted in green allows the user to choose if he/she wants their search to be made in a case-sensitive or insensitive way. On the other hand, the input field highlighted in orange is the one where the user must enter the values that should be searched in the treebank.

The button highlighted in purple shows some options about how the search results should be displayed. For example, it allows the user to indicate whether the part-of-speech tag of each token in every sentence should be displayed on the results page or not.

Note that the whole search input is organized in a way that it forms a sentence (from left to right). This was done this way to improve usability and to decrease the user’s learning curve while he/she is learning how to use the tool.

Complex Searches Users can also build complex search patterns with the use of the logical conditions AND, OR, and NOT. The logical AND and the logical OR are represented by new rows in the search input, while the logical NOT is represented by a value in the selectors highlighted in black in Figure 3.

The AND and OR logical conditions can be added to the search by clicking on any of the buttons labeled *AND* and *OR* highlighted in pink in Figure 3. The

The screenshot shows a search interface with three rows of criteria. Each row has a selector (I want / I don't want), a 'to look for' dropdown, an 'in a' dropdown, a 'case sensitive' dropdown, and a 'way' text input. Between rows are logical operators 'AND' and 'OR'. At the bottom are 'Search treebank' and 'Show options' buttons.

Row	Selector	to look for	in a	case sensitive	way	Logical Operator
1	I want	forms			de novo	AND / OR
2	I don't want	pos tags			[any] ADJ	AND / OR / X
3	I want	pos tags			ADP NOUN	AND / OR / X

Fig. 3. Search input with logical conditions.

logical condition is added below the row that contains the clicked button. To delete a logical AND or logical OR, the user must click on the red button that is on the same row that he/she wants to remove.

Like we mentioned before, the logical NOT is represented by a value in the selectors highlighted in black in Figure 3. This selector has the following two possible values:

- “want”: the row will be evaluated *without* a logical NOT;
- “don’t want”: the row will be evaluated *with* a logical NOT.

To better clarify this manner of searching, we will describe a short example below.

Example: searching with logical conditions Suppose a user wants to search for the following 2-gram in a treebank:

- The first token of the 2-gram has the form “de”;
- The second token of the 2-gram has the form “novo” *AND* its part-of-speech tag is not “ADJ”. Alternatively, the token can have a part-of-speech tag of “NOUN”.

To make this search using UDConcord, users have to fill out their search input like shown in Figure 3. Three rows of input are necessary. The first is the initial one and the other two are for the *AND* and *OR* logical conditions.

In the first row, we select the “forms” option in the second selector and enter “de novo” in the text input. That is because we want to search for two tokens, in that order, with that specific forms. Furthermore, we select the “want” option in the first selector because we want the found tokens to have their forms equal to the ones specified in the text input.

In the second row, which defines the logical AND, we select “pos tags” in the second selector. In the first selector, we choose “don’t want” because we need the found tokens to have their part-of-speech tags different than the ones

specified in the input text. Then, in that input text, we enter “[any] ADJ”. The “[any]” is a special word in our program that signals that a token can have any value. In the case of our example, the first token can have any part-of-speech tag value. Note that the “[any]” is not affected by the “don’t want”. In addition, the “[any]” keyword can also be used to search for skip-grams, by putting it in the place of the token that should be skipped in the query.

Finally, in the third row, which defines the logical OR, we select “pos tag” in the second selector. Furthermore, in the first selector, we choose “want” because we need the found tokens to have their part-of-speech tags equal to the ones specified in the input text. In that input text, we enter “ADP NOUN”. Then, we click on the “Search treebank” to start the search.

Note that there is a specific order of precedence in the evaluation of the logical conditions. First, NOT logical conditions are evaluated, then AND, then OR.

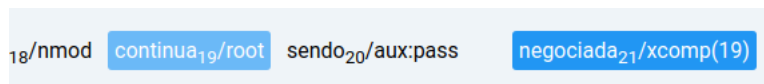
4.2 Displaying the Results

After defining their search parameters and clicking on the “Search treebank” button, the search will be made, and the results are going to be displayed on the screen in the form of a concordance. That means that each sentence with a matching n-gram will be displayed in a row, with the found n-gram centralized and highlighted in blue.

Visualization Options: Displaying Part-of-speech Tags As mentioned in subsection 4.1, the button highlighted in purple (Figure 2) shows some options about how the search results should be displayed. Clicking on it exposes three checkboxes with the following options: *POS Tags*, *Dependency relations*, and *Features*.

These options enable the user to choose a few extra data he/she might want to be displayed on the results page. For example, if he/she checked the box “POS tags”, the part-of-speech tags of every token would be displayed in the search results after a “/” character.

Visualization Options: Displaying Dependency Relations Note that users can also choose to display dependency relations on the results. In that case, these dependency relations would be displayed as shown in Figure 4.



18/nmod continua₁₉/root sendo₂₀/aux:pass negociada₂₁/xcomp(19)

Fig. 4. Display of dependency relations in the results.

The darker blue highlights the matched token/n-gram in the search and the dependent in the dependency relation. The number between parentheses is the token’s head’s id. Before the left parenthesis, there is also the label of the dependency relation. The lighter blue highlights the head of the dependency relation. So, Figure 4 is indicating that there is a dependency relation of type “xcomp” between “negociada” (dependent) and “continua” (head). Also, note that each token has its id subscripted next to it.

4.3 Editing Sentence Annotation

If the user double clicks on a sentence in the results, he/she will be redirected to the corresponding CoNLL-U. There, they will be able to edit the sentence’s annotation data.

The editor is organized in a table-like manner, with each line of the sentence’s CoNLL-U corresponding to one row in the editor. Therefore, a row can have the sentence’s metadata, or token/word annotation – in that case, the row is divided into 10 cells: one for each field defined in the CoNLL-U format. Editing a table cell is the same as editing one value in the CoNLL-U annotation. Currently, UDConcord does not validate the values entered in the cells.

If the user right-clicks on a row, UDConcord will present the following three options:

- Add a row below the one clicked;
- Add a row above the one clicked;
- Delete the clicked row.

It is important to note that if the user removes or adds a row, the ids and the head field values from each row are automatically adjusted.

When the user finished the editing on the CoNLL-U file, he/she just have to press the button labeled “Save changes” (at the bottom right) to save their changes. Then, he/she can go back to the results page by clicking on the button labeled “Go back” (also at the bottom right).

4.4 Downloading the Edited Corpus

UDConcord offers several options to download an uploaded treebank after the user made changes to it. These options can be found after clicking on the arrow inside the blue button labeled “Download” located at the bottom right of the results page:

- “Download treebank (.conllu)”: clicking on it starts the download of the uploaded corpus, with every change made by the user;
- “Download search results (.conllu)”: clicking on it starts the download of the CoNLL-U of every sentence returned by the search. This is great if you want to filter the CoNLL-U of only a few sentences;

- “Download search results (.csv)”: clicking on it starts the download of the search results in a .csv format (not the CoNLL-U, just the sentences themselves);
- “Download search results (.txt)”: clicking on it starts the download of the search results in a .txt format (not the CoNLL-U, just the sentences themselves).

5 Caveats

As we previously mentioned, one of UDConcord’s main goals is to provide an easy-to-use experience to its users. In order to do that, we implemented a query interface that is simple and intuitive. However, this simplicity come with limitations on the kind of queries that can be made with the tool. For example, currently, there is no way to search for head-dependent relations, like a NOUN that is a dependent of the verb “eat” with the “obj” relation. Our tool does not support this kind of query, because its complexity did not allow it to be well translated to our query interface in a simple way. For users that want to make use of a more robust query mechanism that is not covered by UDConcord, we recommend tools like Grew-match and Interrogatório, which allow them to search using standard query languages.

6 Final Remarks

UDConcord was designed to be simple and easy to use. We developed it to ease the task of working with Universal Dependencies-based treebanks. This is especially useful to non-computer savvy users, who might encounter some problems when dealing with the CoNLL-U files.

While simplicity is the tool’s main characteristic, we also focused on providing the necessary features for quality treebank analysis, search, and annotation.

UDConcord is available at <https://udconcord.icmc.usp.br/>. Other related resources and tools may be found at the web portal¹ of the POeTiSA project (which stands for “POrtuguese processing - Towards Syntactic Analysis and parsing”). In this project, UDConcord has been used as support to the construction of the Porttinari treebank for Brazilian Portuguese [7].

Acknowledgements

The authors are grateful to the Center for Artificial Intelligence of the University of São Paulo (C4AI²), sponsored by IBM and FAPESP (grant #2019/07665-4).

¹ <https://sites.google.com/icmc.usp.br/poetisa>

² <http://c4ai.inova.usp.br/>

References

1. Guibon, G., Courtin, M., Gerdes, K., Guillaume, B.: When collaborative treebank curation meets graph grammars. In: Proceedings of The 12th Language Resources and Evaluation Conference. pp. 5293–5302. European Language Resources Association, Marseille, France (May 2020), <https://www.aclweb.org/anthology/2020.lrec-1.651>
2. Guillaume, B.: Graph matching and graph rewriting: GREW tools for corpus exploration, maintenance and conversion. In: Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations. pp. 168–175. Association for Computational Linguistics, Online (Apr 2021). <https://doi.org/10.18653/v1/2021.eacl-demos.21>, <https://aclanthology.org/2021.eacl-demos.21>
3. Heinecke, J.: ConlluEditor: a fully graphical editor for Universal dependencies treebank files. In: Universal Dependencies Workshop 2019. Paris (2019), <https://github.com/Orange-OpenSource/conllueditor/>
4. Martens, S.: Tundra: A web application for treebank search and visualization. In: Proceedings of The Twelfth Workshop on Treebanks and Linguistic Theories (TLT12). pp. 133–144. Association for Computational Linguistics (2013), <http://bultreebank.org/TLT12/TLT12Proceedings.pdf>
5. Nivre, J.: Towards a universal grammar for natural language processing. In: Proceedings of the 16th International Conference on Intelligent Text Processing and Computational Linguistics. pp. 3–16. Springer, Cham, Switzerland (2015)
6. Nivre, J., de Marneffe, M.C., Ginter, F., Hajič, J., Manning, C.D., Pyysalo, S., Schuster, S., Tyers, F., Zeman, D.: Universal dependencies v2: An evergrowing multilingual treebank collection. In: Proceedings of the 12nd International Conference on Language Resources and Evaluation. pp. 4034–4043. European Language Resources Association, Marseille, France (2020)
7. Pardo, T., Duran, M., Lopes, L., Felippo, A., Roman, N., Nunes, M.: Porttinari - a large multi-genre treebank for brazilian portuguese. In: Proceedings of the XIV Symposium in Information and Human Language (STIL). pp. 1–10. Sociedade Brasileira de Computação, Porto Alegre, RS, Brasil (2021). <https://doi.org/10.5753/stil.2021.17778>, <https://sol.sbc.org.br/index.php/stil/article/view/17778>
8. de Souza, E., Freitas, C.: ET: A workstation for querying, editing and evaluating annotated corpora. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 35–41. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic (Nov 2021), <https://aclanthology.org/2021.emnlp-demo.5>