

When Collaborative Treebank Curation Meets Graph Grammars

Gaël Guibon, Marine Courtin, Kim Gerdes, Bruno Guillaume

▶ **To cite this version:**

Gaël Guibon, Marine Courtin, Kim Gerdes, Bruno Guillaume. When Collaborative Treebank Curation Meets Graph Grammars: Arborator With a Grew Back-End. LREC 2020 - 12th Language Resources and Evaluation Conference, May 2020, Marseille, France. hal-03021720

HAL Id: hal-03021720
https://inria.hal.science/hal-03021720

Submitted on 24 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 5291–5300
Marseille, 11–16 May 2020
© European Language Resources Association (ELRA), licensed under CC-BY-NC

When Collaborative Treebank Curation Meets Graph Grammars

Arborator With a Grew Back-End

Gaël Guibon, Marine Courtin, Kim Gerdes, Bruno Guillaume
LLF (CNRS) – Université Paris Diderot & Almanach (Inria),
Almanach (Inria) & LPP (CNRS) – Sorbonne Nouvelle,
Almanach (Inria) & LPP (CNRS) – Sorbonne Nouvelle,
Loria (Inria)
Paris & Nancy, France
gael.guibon@inria.fr, marine.courtin@sorbonne-nouvelle.fr, kim@gerdes.fr, bruno.guillaume@inria.fr

Abstract

In this paper we present Arborator-Grew, a collaborative annotation tool for treebank development. Arborator-Grew combines the features of two preexisting tools: Arborator and Grew. Arborator is a widely used collaborative graphical online dependency treebank annotation tool. Grew is a tool for graph querying and rewriting specialized in structures needed in NLP, i.e. syntactic and semantic dependency trees and graphs. Grew also has an online version, Grew-match, where all Universal Dependencies treebanks in their classical, deep and surface-syntactic flavors can be queried. Arborator-Grew is a complete redevelopment and modernization of Arborator, replacing its own internal database storage by a new Grew API, which adds a powerful query tool to Arborator's existing treebank creation and correction features. This includes complex access control for parallel expert and crowd-sourced annotation, tree comparison visualization, and various exercise modes for teaching and training of annotators. Arborator-Grew opens up new paths of collectively creating, updating, maintaining, and curating syntactic treebanks and semantic graph banks.

Keywords: dependency treebanks, annotation tools, crowd-sourcing, class-sourcing, error-mining, graph banks

1. Introduction

Dependency treebanks have become the standard resource for training syntactic parsers, and substantial efforts have been undertaken to develop large scale and multi-lingual treebanks. The flagship project is certainly Universal Dependencies (UD) (McDonald et al., 2013), which has served as the input to numerous parsers, text generators, and morphological taggers around various shared tasks (Zeman et al., 2018; Mille et al., 2019; McCarthy et al., 2019). The impressive project with more than a hundred treebanks in the same annotation scheme for 90+ languages, combined with great online viewers and query tools have given increased visibility to the project also inside the syntax and typology communities (Croft et al., 2017; Gerdes et al., 2019b).

Yet, for UD as well as for other treebank creation projects, many of the treebanks contain substantial errors and inconsistencies, which can be attributed to three main causes:

1. Many of the UD treebanks are converted from other formats that do not contain all the information needed for a transfer into UD, or the converters are incomplete.

2. Some descriptions in the UD guidelines are underspecified and leave room for different analyses of the same construction, inside a language, a language group, or generally among languages, *cf.* the constantly active UD discussion group on GitHub and also Gerdes and Kahane (2016). Moreover, some UD rules are not well-adapted for specific languages – *e.g.* the discussion of the direction of coordination in Japanese (Kanayama et al., 2018).

3. Some treebank creators do not have sufficient time or competence to provide satisfactory analyses of some of the innumerable syntactic phenomena of their language. Moreover some of the treebanks have been abandoned and do not follow the latest updates of the UD specifications.

Generally speaking, dependency parsing and tagging with recent quantitative NLP tools can overcome, to a certain degree, rarely occurring errors in the training corpus. This is particularly true when making use of word vector representation models trained on massive amount of raw textual data (Devlin et al., 2019; Peters et al., 2018). However, in the use cases of language teaching, the query of counterexamples to syntactic claims, or typological comparative measures of (word order, syntactic relation, construction) distribution tendencies, these errors can influence the results significantly. See for example the differences between treebanks of the same language reported by Chen and Gerdes (2017). It has become essential for the UD project not only to facilitate treebank curation for the treebank maintainer but to find ways to open treebank corrections to a wider audience of linguists and language students.

All the aforementioned treebank annotation errors come essentially in two flavors: occasional slips of attention of the annotator and systematic discrepancies with the desired correct analysis. The former type of problems (occasional slips of attention of the annotator) can be addressed by means of an easy access to "strange" constructions[1] and myriads of annotators who look at these potential issues and who then either directly fix the error or validate the rare construction as being correct. The later type of problems (systematic discrepancies with the desired correct analysis) needs systematic corrections by means of a graph grammar

[1] In the simple sense of being rare or in the sense of not corresponding to the prediction of a parser that has been trained on the rest of the corpus.

and non-regression validation[2]. Examples of this kind of systematic errors are provided by Haverinen et al. (2011) where the authors found that annotators frequently confused direct objects and nominal modifiers, syntactically readily mistakable. But annotators also confused subjects and adjectival modifiers for the surprising reason that the annotation tool's shortcut keys are placed next to one another on the keyboard. These types of error can easily be detected using Grew (Bonfante et al., 2018).

Our new tool Arborator-Grew provides support for the whole process of treebank creation, publication, error-mining, and curation. It is essentially a front-end editor to the Grew graph rewriting system[3] (Guillaume et al., 2012; Bonfante et al., 2018), adding access control, predefined error mining queries, exercises for teaching and annotator training, graphical *diff* tools, and versioning via GitHub. Arborator-Grew is under active development and a first public release is planned during Spring 2020. It attempts to replicate the features of the current version of Arborator (Gerdes, 2013), in particular its class-sourcing tools (Zeldes, 2017), while improving and modernizing queries, error-mining, versioning, and collaborative features. It is the first tool to integrate complex graph querying and treebank annotation software. The advantages of this combination will be discussed in Section 4.

2. Related Work

Quite a few tools exist for dependency treebank development, visualization, and querying, many of which share some of the features of Arborator-Grew. These tools can be divided into three mains groups: dependency visualization tools, dependency annotation tools, and treebank query tools. Without claiming to be exhaustive, we present here the main features of some of these tools, which have led us to the development of Arborator-Grew's specifications.

Dependency visualization. UDAPI (Popel et al., 2017), CoNLL-U viewer[4] and TüNDRA (Martens, 2013) provide interfaces to visualize dependency trees from CoNLL formatted files. Arborator's side project *Arborator-Draft* provides a simple (but fast since D3.js-based[5]) Javascript plugin for webpages that translates any CoNLL data inside <conll> tags into graphical trees[6] giving a lightweight alternative to Arborator. An online version for copy-pasting and visualizing CoNLL files is available on the Arborator homepage[7].

Dependency annotation. These tools are designed for treebank development and allow users to build dependency structures on top of their corpus, without having to resort to manually editing the CoNLL files. Some tools are minimalist, lightweight, and usually offline, others allow for collaborative online annotation of multi-layer treebanks.

The most used tool is probably Brat (Stenetorp et al., 2012). The Brat user can annotate any highlighted span (the user decides where tokens begin and end), which makes it a great tool for (named) entity annotation and chunking, although the annotation is carried out through rather cumbersome dialog boxes. However, just like Arborator, it supports drag and drop of relations between tokens. WebAnno (de Castilho et al., 2016) is a similar tool that shares the visualization front-end with Brat, while modernizing keyboard interactions and back-end, as well as allowing for more web-based project configurations. The visualization uses a multi-line configuration where relations can go across different lines, which can be confusing, but remains necessary as the same format is also used to annotate coreference and other long-distance relations across many sentences. Recent CoNLL-U files need to be converted first in Brat's and WebAnno's internal standoff formats.

Single user online graphical CoNLL file editors include Arborator's Quick online tool[8], Annotatrix (Tyers et al., 2017) (providing Latex export), and the ConlluEditor (Heinecke, 2019). The ConlluEditor is noteworthy for its easy token splitting and joining, its stemma-like horizontal visualization, its integration of UD validation scripts, and its interaction with GitHub versioning.

One last annotation tool worth mentioning in this context is ZombiLingo (Guillaume et al., 2016), a tool for crowd-sourcing of the syntactic annotation process through gamification. Users have to pass rather basic proficiency tests to be allowed to play: they are presented with a sentence at a time, for which they have to determine one single relation, such as finding the subject of a verb, and they can gain points and enter a public wall of fame. The annotations that most players agree on are combined into dependency trees of yet unannotated sentences.

Query tools. The most famous linguistic annotation query tool is arguably Annis (Zeldes et al., 2009). Annis proposes its own query language AQL (ANNIS Query Language) which can handle multi-layer annotations, a graphical query builder, chunk, phrase structure tree, and dependency visualization (the latter using Arborator's visualization Javascript library), integration of sound files, and queries on multiple tokenizations thanks to its stand-off format (Krause et al., 2012). Yet, due to its complexity, Annis requires a non-trivial installation and data-insertion process. AQL is rather verbose, and it is not blazingly fast. Other tools are more specifically designed for queries into single-layer dependency treebanks. One of these tools is Dep_Search from Turku (Luotolahti et al., 2017). It is very lightweight and fast, with a succinct and quite powerful query language based on TGrep (Rohde, 2005), though quite unfamiliar and tricky for users trained on other query languages.

Most of these tools are designed to provide the matching

[2]This can by realized by means of a set of correct target trees that have to be attained by the conversion grammar.

[3]http://grew.fr

[4]https://github.com/rug-compling/conllu-viewer

[5]https://d3js.org/

[6]https://github.com/Arborator/arborator-draft

[7]https://arborator.github.io/live.html

[8]https://arborator.ilpga.fr/q.cgi

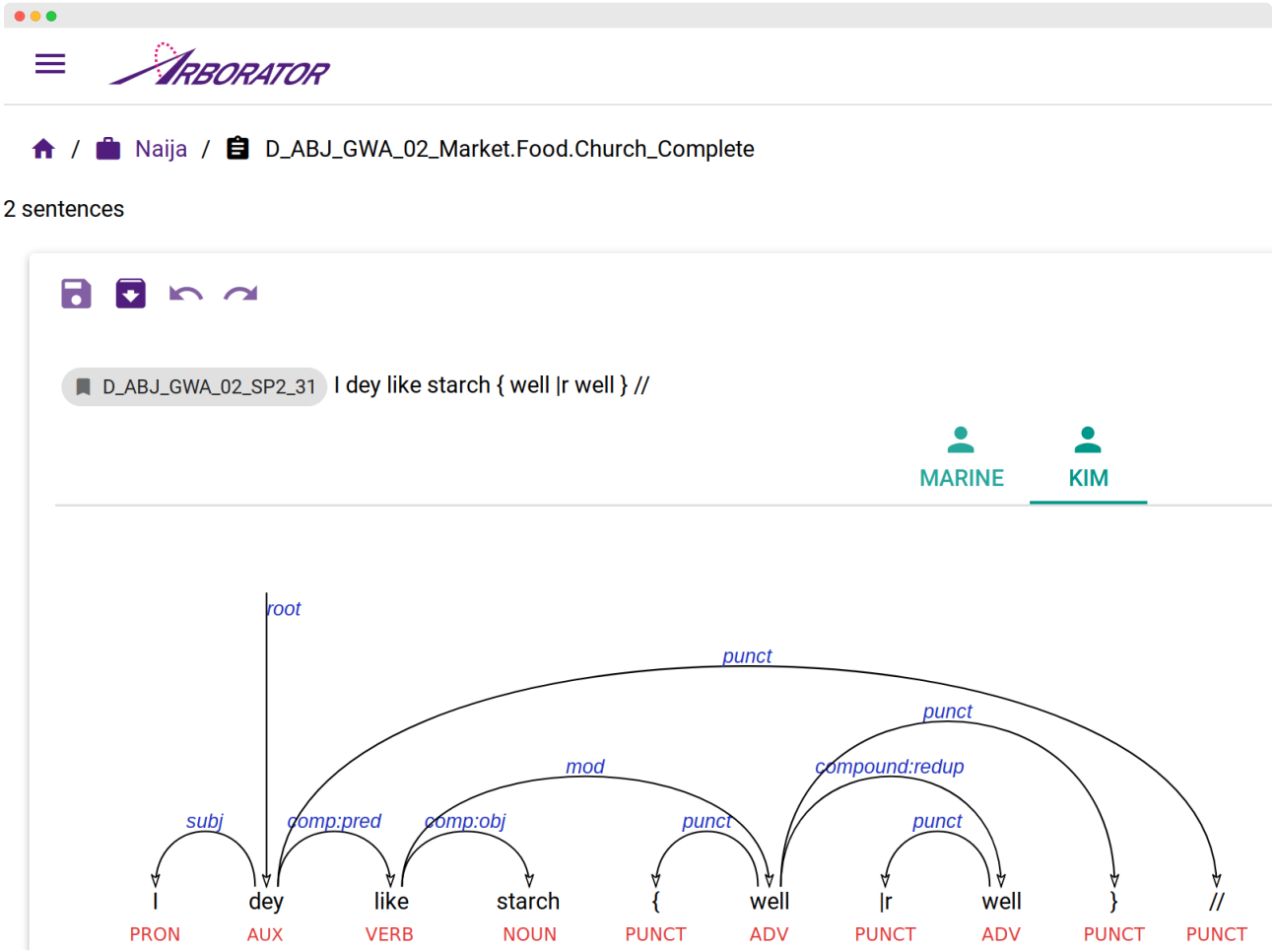


Figure 1: A screenshot of the user interface that gives access to the different annotations of a sentence, with one dependency tree per user. The sentence is drawn from the ongoing treebank annotation project of spoken Naija (Nigerian Pidgin-Créole). The transcribed sentence has a so-called macro-syntactic markup instead of standard punctuation and can roughly be translated by *I really like maize dumplings*.

trees alongside a simple count of matches, but they do not include further statistical data about the query results. Grew-match[9] goes a step beyond that with the possibility to cluster on any of the nodes or edges of the query results. This includes simple clustering of the form for any lemma, thus providing a list of forms, and also the clustering of the relation between any two parts of speech, providing a list of relations that link the two parts of speech. The integration of this feature into Arborator-Grew and its usage will be explained in Section 3.2.

Another remarkable tool that goes a step further is the Trameur (Fleury and Zimina, 2014). It applies corpus linguistics statistical tools to raw corpora, and, in its online version iTrameur[10] also to dependency treebanks. It can therefore show significant over or under-representations of specific sub-trees in one sample compared to another. We intend to study further the possible use of these measures in Arborator-Grew's error-mining tools.

One important question in the design for multi-user annotation systems is the status of tokenization. Brat and WebAnno allow the user to annotate any span of text, most other tools consider that the annotated object is a series of (predefined) tokens. It may seem like a more natural choice to actually annotate texts and to combine the tokenization and syntactic annotation step into one single task. Note however that annotator-based tokenization complicates significantly the computation of inter-annotator agreement, as we jointly observe tokenization and syntactic annotation. And most importantly, tokenization is either trivial and orthography based (i.e. a token is a sequence of letters, possible errors are expressed in the syntactic annotation) or based on lexical and semantic criteria, which makes it a challenging task to reach a satisfying inter-annotator agreement (Farahmand et al., 2015; Savary et al., 2017). Arborator takes a middle stand, making use of its hierarchy of user modes, see Section 4., and allows validators and project owners to modify (delete, add, join, split) tokens, but these changes are then carried out on all trees, whatever the user, of the modified sentence. Such a global modification of a sentence's tokenization can cause other annotators' trees to be disconnected or different from the desired structure. This

[9] http://match.grew.fr

[10] http://www.tal.univ-paris3.fr/trameur/iTrameur/ iTrameur only has a French interface.

behavior is the only exception to the basic Arborator rule which states that users can view other annotators' trees, depending on their access level, but can only create or modify their own tree.[11]

3. Architecture

Arborator-Grew is a complete redevelopment that does not share a line of code with the legacy Arborator, beyond the Python CoNLL-U parser. The legacy Arborator is written in Python 2. It is a simple CGI web page and uses a SQLite[12] database with the FTS4 module for fast text searches. User identification is handled via the quite ancient Login Tools[13], and the front-end runs in Jquery-enhanced Javascript with the rather slow Raphael.js[14] for drawing SVG tree graphs. The new Arborator-Grew consists of three completely separate pieces of software that interact via stateless REST interfaces and follows a Model-View-Controller (MVC) architecture shown in Figure 2.

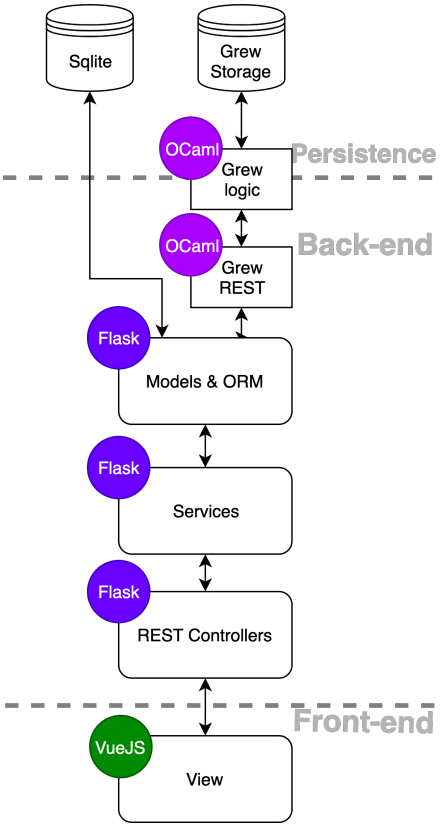


Figure 2: Software architecture of Arborator-Grew

Data Persistence. The database storage relies on the Ocaml[15]-based Grew storage system to which we have added an API accepting json queries. It is possible to run the Grew API on a different server than the main Arborator-Grew, and this is the configuration that we use for our current development setup.

Back-end and User Persistence. The middleman is a Flask[16] application written in Python 3 that uses the Authomatic[17] library for social login. The main task of the Flask application is to control read and write access to the storage back-end, handle software logic through services, and manage different resource routes for the front-end. User information and access rules are persisted in a SQLite database interfaced by an object-relation mapping[18]. The Flask application also pre-mashes some of the replies to make it easier for the front-end to quickly access data, and it keeps in memory frequent queries, mainly for tree comparison in the teaching mode, in order to lower the strain on the Grew server. Note that the Grew API provides CoNLL-U data that the Flask application only passes on to the front-end for visualization. Only meta-information such as the user name and time-stamping is accessed and modified at this stage. Hence, the Flask application represents the Model and Controller in the MVC architecture while the Grew server represents the Data Access Objects related to trees from the Model.

Front-end. Lastly, Arborator-Grew's front-end (Figure 1 shows a screenshot) is written using a Javascript based framework named VueJS[19], which facilitates the development of reactive, modular, and flexible front-end user interfaces, and enjoys rising popularity among web developers. In particular, the Arborator-Grew front-end makes the View part of the MVC independent from the back-end. This facilitates the development of mobile or desktop versions because the same base code can be automatically translated using Node ecosystem packages such as Electron and Cordova. The actual dependency graph is now drawn via the Snap[20] SVG library. The new tree-drawing algorithm has more configuration options. In particular, trees can now take up different heights depending on the complexity of the tree, in order to avoid labels placed too close to one another. Arborator-Grew also adapts better to UD's syntactic relation system that consists of "universal" relations followed by language-specific sub-relations, such as *aux*:*pass*. SUD, the Surface-Syntactic version of UD

[11] If user A views user B's tree and modifies it, the new tree will be saved as the most recent tree of user A, possibly overwriting user A's original tree while leaving user B's tree untouched.

[12] `https://www.sqlite.org/index.html`

[13] `http://www.voidspace.org.uk/python/logintools.html`

[14] `https://github.com/DmitryBaranovskiy/raphael`

[15] `https://ocaml.org`

[16] `https://palletsprojects.com/p/flask/`

[17] `https://authomatic.github.io/authomatic/` For the moment we use Google and GitHub social login, and with Authomatic it is easy to add further social logins to the system. The social login is an important progress for Arborator, as caring for login problems of classes of up to a 100 students can take up considerable time for the teacher.

[18] We used the ORM framework SQLAlchemy: `https://www.sqlalchemy.org/`

[19] We used `https://vuejs.org/` associated to multiple components mostly coming from the Quasar VueJs Framework: `https://quasar.dev/`

[20] `http://snapsvg.io/` Snap is also a faster and lighter redevelopment of Raphael.js by the same authors.

(Gerdes et al., 2019a), further separates deep-syntactic relations, which can give relations such as $comp{:}obj@x$. The combinatorics of universal and other relation names makes it inconvenient to provide the annotator with an exhaustive list of all possible combinations, and Arborator Grew can be configured to show separate choices of universal and secondary relations, see Figure 3.

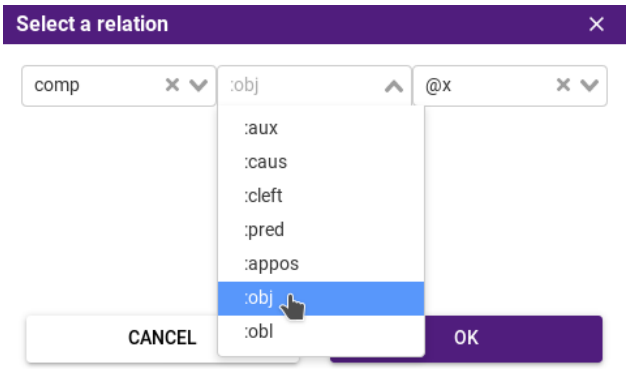


Figure 3: Interface for the relation selection for a SUD project configuration.

4. New Features in Arborator-Grew

The rework of Arborator combined with Grew results in novel ways to build new treebanks and enhance existing annotations. In this section, we focus on two main aspects: classroom collaborative treebank creation (Section 4.1.) and error mining inside an existing treebank using queries (Section 4.2.).

4.1. Class Sourcing

One of the key features of Arborator is collaborative annotation, thus controlling access to the resources must be well-organized. The legacy Arborator was laid out for separate instances on separate servers and it was not designed for multiple projects on the same server with different administrators and different people teaching separate classes. In the new Arborator-Grew, we distinguish the following roles:

1. *Guests*: A project can be public or private. If it is public, guests can browse and query, but not modify the treebank.

2. *Annotators*: Each sample has a list of annotators and validators. Annotators can browse and modify the treebank (modify in the sense that a modified tree is saved under their name). They can keep track of their progress by marking trees and whole samples as "ok". Annotators can be denied access to other annotators' trees, in order to insure independent annotation allowing the computation of inter-annotator agreement.

3. *Validators* of a sample can see all the trees and choose the correct version if the annotators disagree.

4. *Administrators* are assigned during the project creation process. They can change the status of the project, and admit and assign other users (administrators, validators, and annotators). They can also upload new samples to the project, create exercises, and determine the POS and relational tagsets.

5. *SuperAdmins* are determined during the installation of Arborator-Grew, hence they are administrators for the whole system. They can create projects and assign project administrators.

This configuration allows for *Class Sourcing* of the treebank creation, which is a way to merge treebank creation and teaching of academic students, interns, or colleagues willing to learn about syntax. The precision that can be obtained in the setting of class-sourcing has been analyzed in Gerdes (2013), and the actual treebank creation has been demonstrated with the GUM Corpus (Zeldes, 2017), using the legacy version of Arborator for the syntactic component of the corpus.
Annotating actual texts is a great way of studying the syntax of the students' mother tongues as well as of foreign languages where the students have a high degree of proficiency. Arborator, right from its legacy version, was conceived to be used in the classroom, in particular for undergraduate and graduate students.

Exercises modes. Beyond the simple exploration of an existing treebank by means of searches and in-class discussion of various structures, Arborator-Grew allows the configuration of exercises. Exercises have four modes: *graphical feedback* (the student can click on a "check" button and wrong categories or relations are marked in red), *percentage* (the student receives a feedback as a percentage of correct categories and relations, without the indication of where the errors are localized in the sentence), *teacher visible* (the students can see the reference tree, but cannot modify it directly – they have to redraw the tree from an empty annotation), and *no feedback* where only the teacher can receive the student's score.
Teachers need to have an administrator status in the project in order to set up an exercise. For the sample to be used as an exercise, they simply have to choose an exercise mode and determine a reference tree per sentence. Then, the reference tree will be used to provide students with the desired feedback and finally, to compute the students' scores that the teacher can export into a spreadsheet at any time. Arborator's exercises have been tested on various levels and in various countries, and the feedback has been positive from the teachers as well as from students. Syntax exercises seem to feel more like a computer game, in particular the *percentage* exercise mode, where completist students try hard to reach a 100% score.

Treebank construction in the classroom. Training is an essential first step to class-sourcing, and one can go one step further with a class of students pre-trained on a set of exercises: They can be asked to annotate samples where no reference annotation exists. Depending on the number, the level, and the syntactic and linguistic proficiency of the students, various setups have been tried out. In small groups of graduate students, with interns employed for the task, and

among colleagues that want to develop a treebank, the most common configuration is that we provide a first draft of an annotation guide, and we assign a sample to one or two annotators. They annotate the sample and note difficulties that are discussed in a group meeting. Together, the annotation guidelines are updated and specified in order to answer the questions the annotators had.
In the setting of a larger undergraduate class, samples can be distributed to larger sets of students, which allows to compute trees that obtain the highest "votes" by the students in a ROVER-like fashion (Fiscus, 1997). The evaluation of the students has then to be done manually, for example by randomly sampling a few of the student's trees. To automatize this step and in order to avoid students that have been assigned the same sample working together, we use a script (not yet included into Arborator) that combines reference sentences from a gold-standard treebank with yet to be annotated pre-parsed sentences. Optionally, errors can be inserted in both types of trees, using the parser's confusion matrix to make "plausible" errors. For each student an individual set of sentences is prepared, that is given as an assignment. Then, scores can be automatically computed for the student evaluation, but also for a better ROVER vote, where the students' grades are used as a confidence score in the computation (Gerdes, 2013).

4.2. Error Mining

Most treebank developers have already stumbled upon errors in their own, or other people's, treebanks while browsing through the trees. If time allows, the tree has then to be looked up in the latest version of the samples, corrected, and uploaded as a new version. It would be better if the annotator could then check whether similar trees have a similar error. These steps have been quite cumbersome up to now, and this difficulty is the central motivation for the merging of Arborator and Grew.

Pattern matching. The central feature of Grew-match is precisely the query of syntactic patterns in a treebank project. Grew has its own query language which is easy to learn and very powerful. This query language is well-documented[21] and a tutorial is made available on the Grew-match site[22]. The pattern matching system can match subtrees based on forms, lemmas, part-of-speech tags, presence of a morphological feature, value of the morphological feature, incoming and outgoing dependencies (typed or not, enhanced or syntactic), linear order between nodes and any combination of these features, which can result in very complex queries (see an example of a more complex query in Figure 4). It can also filter out these results based on negative patterns (patterns that must not appear in the graph).
The nodes that match the pattern are then highlighted in the trees on the results page. See an example of a query on $comp{:}aux$ relations[23] in the SUD treebank of Old French in Figure 5.
But once the faulty tree has been found, it cannot be modified inside Grew. The next logical step was to integrate Grew as a perfect querying system inside Arborator, thus allowing treebank creators and users to easily find syntactic patterns inside their projects and to directly correct them. In Arborator-Grew, it is now possible to open the Grew-match component (see Figure 6) in a project or in a sample. The result page, containing all the matching trees, possibly from different samples, can then directly be edited and saved.

Clustering the results. Grew also has the ability to cluster the results of a query based on one or several features, thus making it easier to quickly sort through the results. For example, one could look for all the verbs with an object and cluster them based on the part-of-speech of the object. This functionality can be used to build a relation table which summarizes all dependencies within a project, based on the part-of-speech of the governor and dependent. Having this relation table easily accessible is a great way to look for rare structures and potential errors inside a treebank, and to get an overview of the existing structures. In Arborator-Grew, the user can open this table (see Figure 7) and access directly the trees that match the pattern to see if the analysis is correct, and update it if they find an error. Work on this distributional relation table can easily be integrated in a teaching context where each student has to check a different set of relations.
Grew also allows to directly modify samples and whole treebanks by means of a graph rewriting grammar. Access to this feature is planned but not yet realized in Arborator-Grew, as it requires good versioning and feedback in order to avoid the intrusion of unforeseen errors, see Section 6.

5. Distribution

Arborator-Grew is currently accessible through an early version, thus we keep two separate access paths. One for the legacy Arborator[24], another for the new Arborator-Grew[25]. The latter follows the same principle as the former and is distributed in two versions:

- *Aborator Draft*: a simple visualisation Javascript standalone plugin which aims to replace Arborator Quick for a faster visualization tool efficient for large treebanks. Available freely with no account or set up required.
- *Arborator-Grew*: the new server version of Arborator merged with Grew functionalities. It uses an enhanced version of Arborator Draft to handle tree visualization and graphical editing.

In fine we aim at completely replacing the legacy Arborator with the new one. Legacy and reworked Arborator softwares are licenced under the GNU Affero General Public License v3.0 while Grew server is licence under the CeCILL Licence v2.1 [26].

[21] http://grew.fr/pattern/
[22] http://match.grew.fr/
[23] Grew query: pattern { GOV -[comp:aux]->DEP }

[24] https://github.com/Arborator/arborator-server
[25] https://arborator.github.io/arboratorgrew.html
[26] http://cecill.info/licences/Licence_CeCILL_V2-en.html

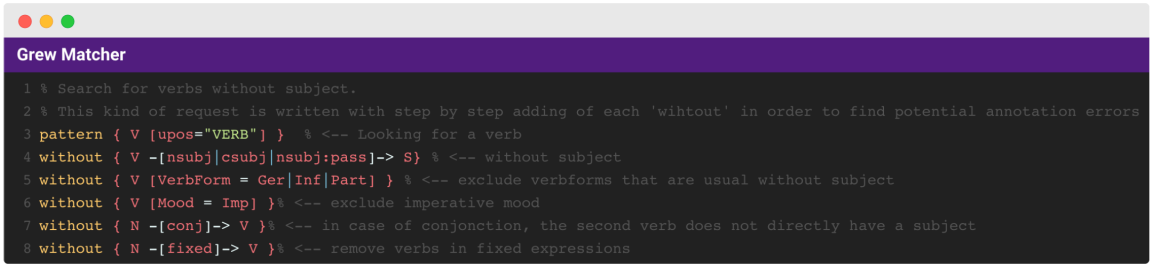


Figure 4: Pattern to look for potential errors on verbs without subjects

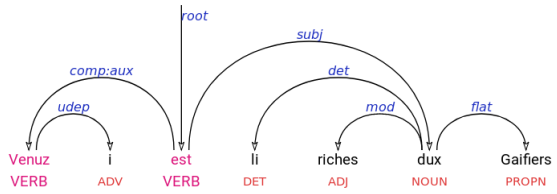


Figure 5: Search result querying a *comp:aux* relation, with pink highlighting of the governor and the dependent. The sentence from the Old French text *La Chanson de Roland* 'The Song of Roland' (1040 - 1115) can be translated by *The rich duke Gaifier has arrived.*"

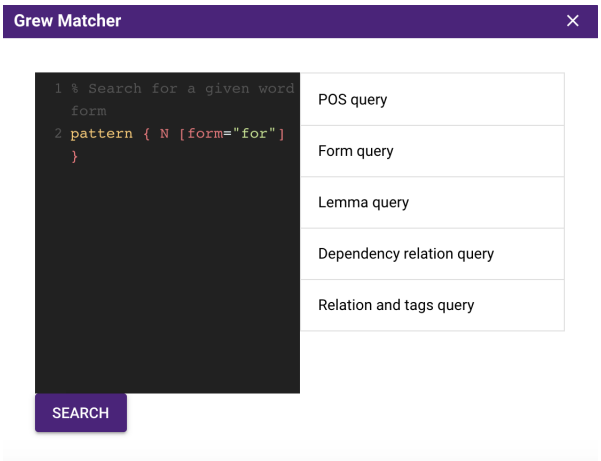


Figure 6: Grew-match integration component, with example queries on the right and automatic idiosyncratic highlighting of the Grew query language.

6. Future Work

The development of Arborator-Grew is currently under heavy development, and we are planning on integrating other features that have been suggested by colleagues working on treebank development.

In particular, we are performing tests to switch the ongoing Naija[27] and Old French[28] treebank development projects to the new Arborator-Grew.

It will also be possible to integrate Grew rewriting grammars in Arborator-Grew where changes are made across a sample or a whole project. This would allow the parallel development of two versions of a treebank which are transferable into one another by rewriting rules. For the development of SUD and UD grammars, where Grew rewriting grammars exist in both directions, annotators can then choose in which view they would like to annotate and propose corrections – the other treebank version being just a click away.

One point that has become very apparent is the need for a better handling of versioning, to allow treebank developers to return to previous states of the annotation, should they need it. The graphical Javascript editor has an undo functionality that does not persist on reload of the editing page. In order to integrate versioning we plan to rely on the GitHub social login integration: A user logged in via GitHub will be able to commit and push a version to his or her GitHub repository. Reverting to previous versions can then be done directly on the GitHub page, making use of the convenient GitHub user interface and also possibly external tools that the user may want to apply to the CoNLL files. At any point, the user will be able to pull the latest version from the repository into Arborator-Grew.

To facilitate error-mining and treebank validation, we are also planning to integrate a validation script that would describe forbidden patterns. This practice has now become part of the Universal Dependencies project, where all treebanks must pass through a validation script to be accepted in the new releases that occur every 6 months, so as to maintain the overall quality of the annotation.

Finally, thanks to the modular architecture of Arborator-Grew with the separation of the front-end from the two (Flask and Grew) REST server engines, laid out in Section 3., it is straightforward to create a mobile version of the front-end part, sharing most of the features with the current desktop version.

Acknowledgements

This work was made possible thanks to the NaijaSynCor French project ANR-16-CE27-0007 (2017-2020) directed by Bernard Caron. The Grew API is hosted in the CPER LCHN (Contrat de Plan État-Région - Langues, Connaissances et Humanités Numériques) infrastructure.

Bibliographical References

Bonfante, G., Guillaume, B., and Perrier, G. (2018). *Application of Graph Rewriting to Natural Language Pro-*

[27] NaijaSynCor is a French national project on treebank development of spoken Naija, the Nigerian Pidgin-Créole with about 100 million speakers (Caron et al., 2019)

[28] Profiterole is another French national project for the diachronic treebank development of Old French (Regnault et al., 2019)

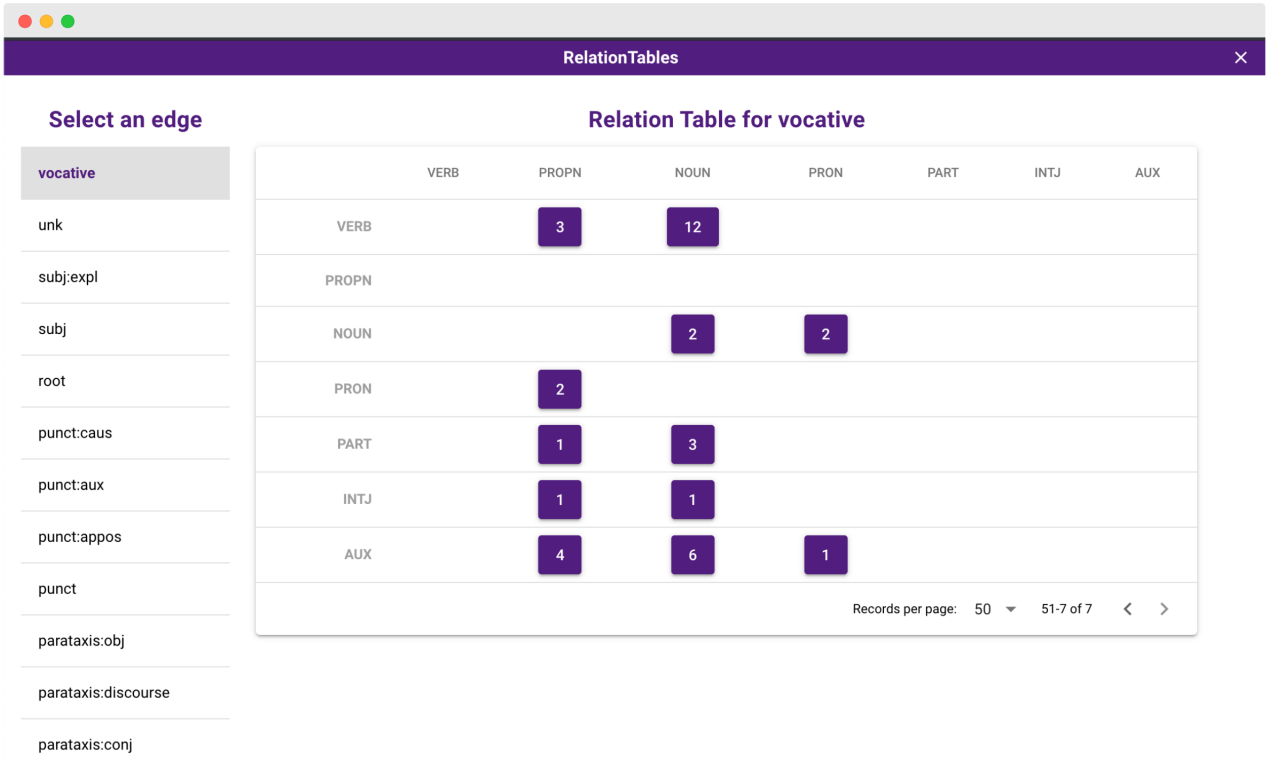


Figure 7: Relation table showing a count of all occurrences of *vocative* relations between a governor (category to the left) and its dependent (category on top) based on their respective parts-of-speech. Here the table query searches in the annotator's own trees; alternatively, the most recent accessible trees can be taken into account. The annotator can directly click to visualize, for example, the two pronouns that have a proper noun as a vocative dependent, in order to verify the two corresponding trees. If a tree contains an error, it can directly be corrected. If the tree is correct, it can be marked as validated, which is saved and indicated in the table during a future query.

cessing. Wiley Online Library.

Caron, B., Courtin, M., Gerdes, K., and Kahane, S. (2019). A surface-syntactic ud treebank for naija. In *TLT 2019 - 18th International Workshop on Treebanks and Linguistic Theories, Aug 2019, Paris, France*.

Chen, X. and Gerdes, K. (2017). Classifying languages by dependency structure. typologies of delexicalized universal dependency treebanks. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017), September 18-20, 2017, Università Di Pisa, Italy*, number 139, pages 54–63. Linköping University Electronic Press.

Croft, W., Nordquist, D., Looney, K., and Regan, M. (2017). Linguistic typology meets universal dependencies. In *Proceedings of the 15th International Workshop on Treebanks and Linguistic Theories (TLT15), Bloomington, IN, USA*, pages 63–75.

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

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Farahmand, M., Smith, A., and Nivre, J. (2015). A multiword expression data set: Annotating non-compositionality and conventionalization for english noun compounds. In *Proceedings of the 11th Workshop on Multiword Expressions*, pages 29–33.

Fiscus, J. G. (1997). A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (rover). In *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pages 347–354. IEEE.

Fleury, S. and Zimina, M. (2014). Trameur: A framework for annotated text corpora exploration. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 57–61.

Gerdes, K. and Kahane, S. (2016). Dependency annotation choices: Assessing theoretical and practical issues of universal dependencies. In *Proceedings of the 10th Linguistic Annotation Workshop held in conjunction with ACL 2016 (LAW-X 2016)*, pages 131–140.

Gerdes, K., Guillaume, B., Kahane, S., and Perrier, G. (2019a). Improving surface-syntactic universal dependencies (sud): Surface-syntactic relations and deep syn-

tactic features. In *Proceedings of the Universal Dependencies Workshop (UDW), SyntaxFest, Paris.*

Gerdes, K., Kahane, S., and Chen, X. (2019b). Rediscovering greenberg's word order universals in ud. In *Proceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest 2019)*, pages 124–131.

Gerdes, K. (2013). Collaborative dependency annotation. In *Proceedings of the second international conference on dependency linguistics (DepLing 2013)*, pages 88–97.

Guillaume, B., Bonfante, G., Masson, P., Morey, M., and Perrier, G. (2012). Grew: un outil de réécriture de graphes pour le tal (grew: a graph rewriting tool for nlp)[in french]. In *Proceedings of the Joint Conference JEP-TALN-RECITAL 2012, volume 5: Software Demonstrations*, pages 1–2.

Guillaume, B., Fort, K., and Lefèbvre, N. (2016). Crowdsourcing Complex Language Resources: Playing to Annotate Dependency Syntax. In *International Conference on Computational Linguistics (COLING)*, Proceedings of the 26th International Conference on Computational Linguistics (COLING), Osaka, Japan, December.

Haverinen, K., Ginter, F., Laippala, V., Kohonen, S., Viljanen, T., Nyblom, J., and Salakoski, T. (2011). A dependency-based analysis of treebank annotation errors. In *In Proceedings of Depling'11.*, pages 47–61.

Heinecke, J. (2019). Conllueditor: a fully graphical editor for universal dependencies treebank files. In *Proceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest 2019)*, pages 87–93.

Kanayama, H., Han, N.-R., Asahara, M., Hwang, J. D., Miyao, Y., Choi, J. D., and Matsumoto, Y. (2018). Coordinate structures in universal dependencies for head-final languages. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 75–84.

Krause, T., Lüdeling, A., Odebrecht, C., and Zeldes, A. (2012). Multiple tokenizations in a diachronic corpus. In *Exploring Ancient Languages through Corpora Conference (EALC)*, volume 14.

Luotolahti, J., Kanerva, J., and Ginter, F. (2017). dep_search: Efficient search tool for large dependency parsebanks. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 255–258.

Martens, S. (2013). Tündra: A web application for treebank search and visualization. In *The Twelfth Workshop on Treebanks and Linguistic Theories (TLT12)*, page 133.

McCarthy, A. D., Vylomova, E., Wu, S., Malaviya, C., Wolf-Sonkin, L., Nicolai, G., Kirov, C., Silfverberg, M., Mielke, S. J., Heinz, J., et al. (2019). The sigmorphon 2019 shared task: Morphological analysis in context and cross-lingual transfer for inflection. In *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 229–244.

McDonald, R., Nivre, J., Quirmbach-Brundage, Y., Goldberg, Y., Das, D., Ganchev, K., Hall, K., Petrov, S., Zhang, H., Täckström, O., Bedini, C., Bertomeu Castelló, N., and Lee, J. (2013). Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria, August. Association for Computational Linguistics.

Mille, S., Belz, A., Bohnet, B., Graham, Y., and Wanner, L. (2019). The second multilingual surface realisation shared task (SR'19): Overview and evaluation results. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*, pages 1–17, Hong Kong, China, November. Association for Computational Linguistics.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June. Association for Computational Linguistics.

Popel, M., Žabokrtský, Z., and Vojtek, M. (2017). Udapi: Universal API for universal dependencies. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 96–101, Gothenburg, Sweden, May. Association for Computational Linguistics.

Regnault, M., Prévost, S., and de la Clergerie, É. V. (2019). Challenges of language change and variation: towards an extended treebank of medieval french. In *TLT 2019 - 18th International Workshop on Treebanks and Linguistic Theories, Aug 2019, Paris, France*.

Rohde, D. L. (2005). Tgrep2 user manual. *Unpublished manuscript* `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/cmt-55/OldFiles/lti/Courses/722/Spring-08/Penn-tbank/Tgrep2/tgrep2{_}manual.pdf`.

Savary, A., Ramisch, C., Cordeiro, S., Sangati, F., Vincze, V., QasemiZadeh, B., Candito, M., Cap, F., Giouli, V., and Stoyanova, I. (2017). The parseme shared task on automatic identification of verbal multiword expressions. In *MWE2017 - Proceedings of the 13th Workshop on Multiword Expressions , Apr 2017, Valencia, Spain.*

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*, pages 102–107. Association for Computational Linguistics.

Tyers, F. M., Sheyanova, M., and Washington, J. N. (2017). Ud annotatrix: an annotation tool for universal dependencies. In *Proceedings Of The 16th International Workshop On Treebanks And Linguistic Theories.*

Zeldes, A., Lüdeling, A., Ritz, J., and Chiarcos, C. (2009). Annis: A search tool for multi-layer annotated corpora. In *In: Proceedings of Corpus Linguistics 2009, Liverpool, July 20-23, 2009.* Humboldt-Universität zu Berlin, Philosophische Fakultät II.

Zeldes, A. (2017). The gum corpus: creating multilayer resources in the classroom. *Language Resources and Evaluation*, 51(3):581–612.

Zeman, D., Hajič, J., Popel, M., Potthast, M., Straka, M., Ginter, F., Nivre, J., and Petrov, S. (2018). CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium, October. Association for Computational Linguistics.