Collaborative Dependency Annotation

Kim Gerdes
Sorbonne Nouvelle
ILPGA, LPP (CNRS)
Paris, France
kim@gerdes.fr

Abstract

This paper presents the Arborator, an online tool for collaborative dependency annotation together with a case study of crowdsourcing in a pedagogical university context. In greater detail, we explore what generally distinguishes dependency annotation tools from phrase structure annotation tools and we introduce existing tools for dependency annotation as well as the distinctive features and design choices of our tool. Finally we show how to setup a crowdsourced dependency annotation experiment as an exercise for university students. We explore constraints, results, and conclusions to draw.

1 Introduction

The importance of treebanks in today's data-driven linguistics cannot be overrated. All data-driven approaches to syntax require gold-standard annotations, and the need for (possibly machine-aided) hand annotation tools is more urgent than ever, as researchers want to go beyond the eternal Penn Treebank derivatives, because of interests in different languages, annotation levels, and theoretical backgrounds underlying the research.

In recent years, dependency treebanks have become the near-standard representation of annotation schemes in computational linguistics. However, the inherently non-local structure of dependencies make graphical annotation tools more difficult to develop and commonly less intuitive to use.

This paper presents an online annotation tool named Arborator, its features, and how it can be used in an educational surrounding at the same time for pedagogical purposes as well as with the goal to develop high-quality dependency treebanks.

2 Context

Even though a vast majority of dependency links are projective, even in so-called free word order languages, one of the major advantages of dependencies is not to presuppose the structure to have certain properties, like being projective. Phrase structure, on the contrary, is based on the underlying assumption of a coincidence between word order and government; contrary cases have to be taken care of by means of "traces" and "movements" (Gerdes 2005). Dependencies can thus represent more abstract relations, closer to semantics, which is arguably the main reason for today's success of dependency in NLP.

On the annotation level, however, dependency is a notably harder notion to handle than constituency. This holds for the file format because phrases can very easily be represented by simple bracketing or elaborated versions of bracketing like XML; dependency needs to separate tokens and links, the links referring back to the token objects. But this also holds for the manual annotation process as many tools exist for the exploration and editing of "tree"-like structures similar to "file/folder structures" – and dependency is somehow orthogonal to this kind of structure.

2.1 Existing tools

Still today, most dependency treebanks are derived from phrase structures by means of rule-based or statistical transformation of phrase structure. The manual quality control occurred on the phrase structure level with appropriate tools. For very small treebanks, some hand-woven approaches are still around (Chen et al. 2011), using for example a simple spreadsheet for annotation.

Only few dependency treebanks are directly constructed as such by use of well-adapted tools, in this section we will give a short overview over the existing graphical tools:

Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013), pages 88–97,
Prague, August 27–30, 2013.

The first tool that included dependency annotation was probably Annotate (Plaehn & Brants 2000), used for the development of the Tiger corpus. It applies Tiger's mixed syntactic structure: Labeled constituents with functionally labeled edges and crossing branches for non-projective structures.

At about the same time appeared the StrEd (Structure Editor, Boguslavsky et al. 2000), also an offline tool meant to facilitate the manual correction of already pre-annotated data. Its graphic representation has the particularity that each token is on a separate line, similar to the common CoNLL format, and the dependency tree is constructed on these tokens, thus turned by 90 degrees compared to more common representations with the root on top. To our knowledge, this is the first tool to use drag-and-drop creation of dependency links.

TrEd, the Tree Editor from Prague is an offline tool written in perl that helped to create and correct the Prague Dependency Treebank (Hajič et al. 2001, Hajič 2005) as well as other treebanks for other languages like English (Rambow et al. 2002) or more recently Persian (Seraji & Nivre 2012) . It includes an interface with a valency lexicon in order to keep the annotations coherent and scripting possibility for batch processing. Moreover, it was probably the first tool that includes visual comparisons between two annotators' trees, including the possibility to choose the correct structure.

NotaBene, developed by Mazziotta (2010) is an open-source (GPL) off-line tool written in Python that presents the above-mentioned file manager type interface. Elements are tokenized when entering data in the tool, but all other forms of automization are explicitely excluded. The tool includes sophisticasted feature handling, tree comparison and it follows RDF standards to capture multi-layer annotations. NotaBene is principally used in an ongoing annotation project of Ancient French.

DepAnn, written by Tuomo Kakkonen 2006 is another offline dependency annotation tool written in java. It can represent and modify the dependency representation of Tiger-XML. It includes a consistency checker and comparison of trees.

Other dependency treebank like the Danish Dependency Treebank, the Alpino Treebank (Van der Beek et al. 2002) or the Turin University Treebank have been elaborated with the help of bootstrapping approaches in a command line interfaces and special dependency tree viewers that allow for faster choices between different automatic parses of the same sentence.

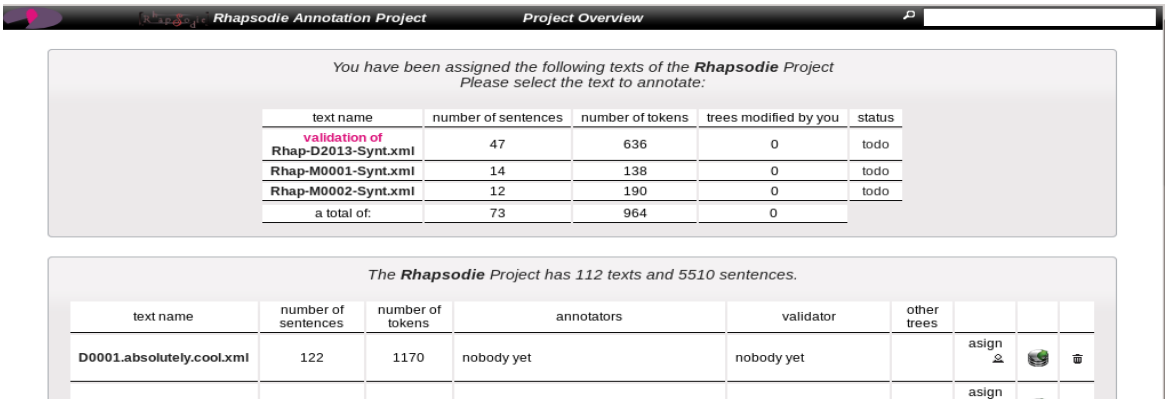
More recently, MATE, a graph transduction workbench (Bohnet et al. 2000; Bohnet, 2006) has been used for the development of multi-stratal corpora in Meaning-Text style (Mel'cuk 1986) of Spanish and, with smaller scope, other European languages (Mille et al. 2009). MATE is written in java and includes a graphical editor for graph structures.

The most sophisticated tool and the closest in design to the Arborator is without any doubt the very recently presented tool “Brat” (Stenetorp, et al. 2012). Like the Arborator, Brat is a web-based application using SVG that allows for graphic drag-and-drop dependency-centric multi-user annotation of text corpora. It also has comparable user management, annotation comparison, Unicode support, and import and export capacities. Contrarily to the Arborator, it is not sentence-based, text appears continuously in multi-line representation, and Brat thus allows for the annotation of intra-phrase relations like co-reference and discourse annotation. Also the segments are not fixed and any continuous chain of letters can be marked and then linked to other parts. Moreover it contains a constraint language that allows for on the fly checking of annotations. The Arborator's search and concordancer features seem to be slightly more developed as a search for specific feature is possible and Brat only includes plain text search. Also the Arborator's corpus distribution and user management seems to be more adapted to “uncooperative” surroundings like the classroom where it is important that the annotators and validators access only the texts that have been assigned to them. These features will be exposed in greater detail in the subsequent sections.

The only other web-based tool that we are aware of is EasyRef (De La Clergerie 2008). EasyRef has a constituent based representation even for dependencies: They are represented as (continuous) segments with a function label. This tool is designed for human evaluation of parser performance which makes it the only other tool including techniques for voting systems (see section 4).

3 Design of the Arborator

The Arborator has been developed over several years in a two-fold perspective: It was needed for the annotation of transcribed spoken language in the Rhapsodie treebank project (Gerdes et. al.



2012), and it is today used as a pedagogical and crowdsourcing tool in various universities. A description of such an experiment constitutes section 4 of the present paper.

This implies the following different design choices:

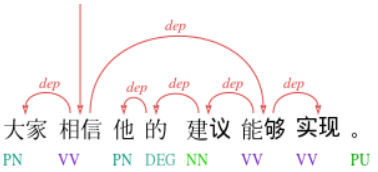
- Zero setup: The tool must run on any computer without any difficult adaptation or installation procedure.
- Central storage of texts and annotations.
- Multi-audience interface: For professional annotators, it needs to include numerous keyboard short cuts for all common annotation tasks, and for starters, the annotation process has to be graphical and self-explanatory.

These points exclude most existing tools and imply the development of a web-based application that does not use any plugins but runs directly in a standard-compliant browser. The graphical nature of the data including arrows forces us beyond simple HTML to an SVG representation of the corpus.

The Arborator can be used for the correction of automatically (or, less commonly, manually) pre-annotated corpora or for the creation of tree structures from scratch. Every token can depend on one (or more) governors and can have simple features attached to them. The choice of features to be shown (and to be modifiable) directly under the token is configurable, the most common ones being of course the syntactic category (POS) and the token's lemma.

Other technical design choices of the Arborator include:

- Development in Python with an underlying Sqlite database with client-side interactions in Javascript (Jquery).
- Runs on any Python-CGI capable Apache webserver.
- Optimized for the Firefox browser but runs reasonably well in other SVG capable browsers.
- Multi-level user hierarchy: site administrator, corpus administrator, validator, assigned annotator, visitor.
- The appearance of the dependency structures is highly configurable in simple configuration files.
- Of course, the Arborator is fully Unicode capable with non-ASCII characters being allowed in sentences, annotation schemes, and login names:



- The design choice of keeping the sentences "readable" with tokens being juxtaposed horizontally is debatable. The alternative, stemma-like structures like those used in the TrEd from Prague, makes the hierarchical structure of the trees more visible, whereas our choice emphasizes the linear sentence structure. We believe that this makes it easier for the annotator to understand the sentence and then to capture the sentence's syntactic structure. But of course, in this matter, beauty is in the eye of the beholder.

The Arborator is employed in different universities for annotation tasks, the main site being http://arborator.ilpga.fr – the main page also provides links to tutorial pages and the source code on Launchpad. The Arborator is distributed under the APGL license, the standard open-source license for server software.

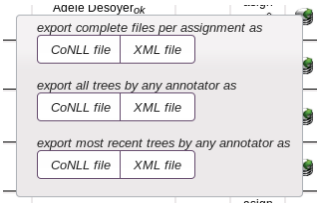
3.1 The user experience

Before using the Arborator, the user has to create an account with email verification. The log-on brings him to the project page containing different option depending on the user level of the user. The normal annotator finds on top of the page the texts that have been assigned to him either as a validator or as an annotator[1]. Each text (and also each sentence) has a changeable status, which allows the annotator to indicate to the validator that the annotation process is terminated
The center part of the project page contains a table with all the texts of the project. The administrator of the project can

- attribute any text to a user's annotation or

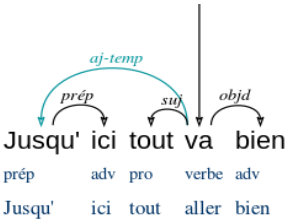
validation tasks,

- export the data in multiple formats and



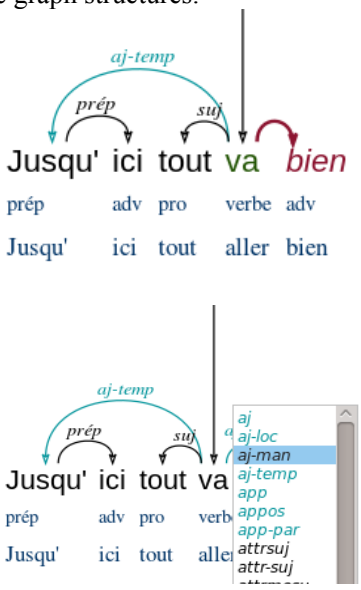
configurations

- add whole texts to the project, plain texts or pre-parsed data in CoNLL or Rhapsodie-XML format.
- Check the consistency of the annotation by obtaining tables of frequency distribution of features and 2-node connected sub-graphs of the dependency graph.
- Obtain an overview of each annotator's progress.



A click on the text name brings the user to the online editor. Depending on his role and the chosen setup of the annotation project, he will see only the words with his own annotation (if present), the standard annotation (for example the pre-parsed structures), or a list of all possible trees for each sentence.

Each token can be dragged and dropped on another token, thus creating a link in this direction. A context menu opens and the user has to choose the corresponding function name (the list of functions is set in the project configuration). When holding the *shift* key down when choosing the function name, the governor is added to the existing governor, allowing thus for the creation of cyclic graph structures.[2]



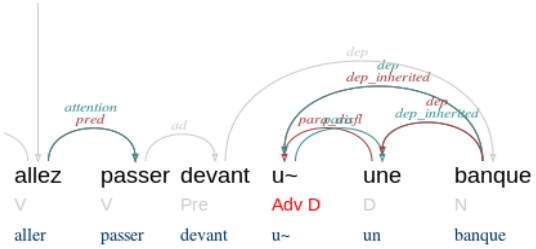
Equally, the shown features can be modified by means of a context menu that opens when the features are clicked upon. A double-click opens a table of other features, including, for administrators, the possibility to modify, add, or erase tokens.

[1] There are various setup options available to control the visibility of different annotations, but in the most basic configuration, the annotator only sees his own trees and the validator can see the trees of all annotators of the given text which allows her to compare between annotations and choose the correct tree.

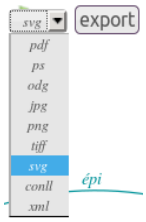
[2] Some analyses of coordination or of relative phrases suppose double governors (for example because the relative pronoun is thought to play the role of the pronoun inside the relative clause and of the complementizer heading the relative phrase). Similarly, cycles have been proposed for the syntactic analysis of collocations and underspecified PP-attachment (Gerdes & Kahane 2011)

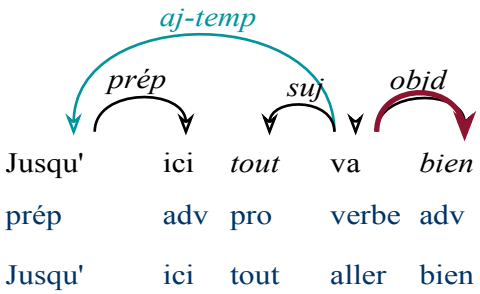
All modifications are undoable (during the annotation session – the Arborator does not yet provide automatic versioning) and the whole annotation (or correction) process can be done exclusively by means of keyboard shortcuts, without touching the mouse, which is often faster for experienced users.

If there is more than one tree visible to a user, he can graphically compare any set of annotations. The resulting graph shows in color the differences and grays out what is in common between the chosen analyses:



Each tree can individually be exported in the following formats. This allows for an easy integration of high-quality vectorial images for publications: SVG (Scalable Vector Graphics), PDF, PS, ODG, JPG, PNG, TIFF, CONLL (tab-separated text table), and XML (an idiosyncratic stand-up format that allows for the linking of the syntactic transcription to sound files)





4 The experiment

The second part of this paper addresses the question of how good the dependency annotation of non-professional annotators can become, if we use a rover, i.e. a voting system (Fiscus 1997), to establish the best annotation among a series of annotations produced by semi-trained students.

This is interesting as many linguistic departments lack resources to train and pay professional annotators, but don't lack students with the desire to learn syntactic analysis.

4.1 Gaudium ex cathedra

Also from a pedagogical point of view, the use of a collaborative online annotation tool has many advantages:

First, the students are often taught in quite large classes and it is impossible for the teachers to systematically correct exercises composed of any larger amount of annotations. The Arborator allows for different types of exercises:

1. the gold-standard annotation is completely visible to the students – they can discover the structures.
2. an incomplete structure is left visible, but the gold-standard remains invisible. The teacher can thus oblige the student to complete parts of an analysis that was the subject of the current class.
3. The annotation is invisible to students, but scoring is public (so students can update their annotation until they hit 100%).
4. equal to mode 2 but the location of the errors are indicated on the sentence which guides the student more quickly to the right annotation.

The teacher, on the other hand can, with little effort, create interactive playful syntactic training sessions, and obtain, for free, a completely automatic list with student evaluations, thus forcing the students to work regularly. Of course, any e-learning environment allows for the creation of multiple choice tests, but it is difficult to make them as interesting and well-adapted to linguistic analysis.

Secondly, the task of annotation of raw data forces teacher and student to abandon easy handcrafted example sentences and allows them to face the cruel realities of language. When collaborative corpus annotation is taken as the main goal of a class, the questions and debates that come up in the classroom are of much more exiting and motivated nature than conventional teaching of syntax.

4.2 Context

The experiment was carried out with a class on corpus linguistics taught to about 60 3rd year linguistics majors in a French university, about 75% of which have French as their mother tongue. Only 3 main classes and 3 tutorial session in smaller groups were held on the subject of this project. Nearly all those students had have other classes on syntax, one of which was taught one year earlier by the same teacher specifically on dependency syntax, using very similar notational conventions as those used in the annotation guide for the experiment. The annotation guide was online and contained many concrete examples, also including the supposed analysis of language idiosyncrasies such as dates, numbers and punctuation, that are frequently encountered in Wikipedia and journalistic examples making up the essential parts of the texts to be analyzed.

4.3 Annotation task distribution

We have two sets of sentences:

- the mini gold-standard annotated by the researcher
- the non-annotated sentences, considered as unlimited

The goal is to distribute the sentence to the students in a just and reasonable manner. As online annotation does not provide the possibility to control the context in which the student annotate, it is important to make it difficult to blindly copy annotations from one student to the other (although theoretically cooperation of students during the annotation process could be useful to obtain better analyses of the syntactic structure). The system can distribute the sentences of the texts into task sets using the following parameters:

- *t*, total number of tokens per student to annotate (rounding up or down in order to distribute complete sentences)
- *g*, number of sentences from the pre-annotated mini gold-standard to mix into the student's task (to allow for evaluation)
- *n*, number of annotations per sentence (taken from the non-annotated sentences)
- *p*, percentage of sentences that can be equal from one task set to the other.[3]

The Arborator comes with a script that optimizes this distribution.

4.4 Setup

The students were presented with 48 sentences, mostly taken from two French Wikipedia articles and some constructed sentences that contain phenomena discussed in the class. The average length of 24.7 tokens per sentence reflects the "real world origin" of most sentences, very different from common example sentences from syntax classes or textbooks.

The tokenization is simply sign-based and was done automatically.[4] All sentences were also annotated by the teachers of the class. This is, of course, only necessary for this experimental

[3] If *p*=100%, *n* students receive equal tasks. When *p* decreases, for example to 50%, the first student will share 50% of her sentences with the second student, and another 50% with the third student, and so on.
Note that the total number of students is not part of these parameters, because in the natural setting of a class taught to a large number of students, many inscribed students will drop the class, and new students appear. Only when a student creates her account on the Arborator, her task is prepared. This minimizes the number of sentences that do not obtain *n* annotations in the end of the project.

[4] A more sophisticated tokenization would have been of use for a few special cases of French syntax. The most problematic case is the token *des* that can be the contraction of *de* 'of' and *les* 'the',or it can just be the plural article *des*. Other problematic cases include *parce que* 'because' and *c'est-à-dire* 'which means'. A production environment would in any case start of with the output of a parser that should do better on tokenization.

setup, and if the tool is to be used in an production environment, only a small number of sentences need to be annotated by the researchers, the rest will be done by the "crowd" of students.

This experiment is only concerned with simple dependency structures. The labels, i.e. the syntactic functions, are left aside for future research. One reason for this is that the results on functions appear, on a quick glance, much worse than the government structure. This is partly due to the fact that the students were told that the government structure is more central to the exercise than the choice of function; and the government structure was discussed in greater detail in the classes. Another reason is that the set of syntactic functions was unnecessarily (and uncommonly) large, including distinctions like locative, manner, temporal, and other adjuncts, etc.

We only kept annotations when 80% of the words were annotated (i.e. had a governor) and, in order to get reasonably good evaluations, we only kept the annotation of students who at least annotated 5 sentences. This left us with 42 student annotators. Using the evaluation based on all sentences, the quality of the dependency annotators ranges from 64% to 90% of correct government relations (F-score), the average being 79%. How many sentences do we have to take into account in the evaluation if we want to keep similarly precise evaluation scores of the student, needed for the rover? Interestingly, the student evaluation varies very little if we base it on the first half of the corpus only (less than 1% in average), the quality of the annotation is better (80%) on the first half, probably due to symptoms of fatigue of the annotators and discussions in class of problems the students encountered. If we decrease further the number of sentences that we base our evaluation on, the evaluation averages continue to grow, but the students' F-score decreases quickly.

Nr of sentences used for evaluation	48	24 (½)	12 (¼)	6 (1/8)	1 (1/48)
Min F-score	64%	67%	70%	73%	63%
Max F-score	90%	90%	90%	90%	100%
Average F-score	79%	80%	81%	83%	87%
Average difference from complete F-scores	0	1.2%	1.8%	3.5%	9.1%

Note that these F-scores are computed in the Arborator and can be exported and thus used directly for grading students. Let's now see how these scores are used in the voting system.

4.5 How many sentences for student evaluation?

When using one part of the trees for evaluation of the students, and constructing an optimal tree on the remaining sentences we obtain the following results. At the present state we always split into a first part for computing the students' scores and a second part which are the remaining sentences. Successive studies will try different jackknifing techniques.

The construction of an optimal tree is slightly complicated by the graph structure of the analysis, i.e. the possibility of double governors, as explained above. So the first step of the different voting systems is to decide on the number of governors, 1 most of the time, but sometimes 0 (errors in segmentation) or 2 (only relative pronouns with our annotation guidelines for French).

The *Scoring* voting system works as follows:

For every node, every proposal of a governor node gets the score the annotator obtained in the evaluation. Then the governor (or the two governors, if the first vote decided on two governors) with the highest score is chosen for the tree. Note that this does not include explicit coherence tests (like non-circularity etc.) but we have not discovered any circular tree with our data.

In this first version, only students can take part in the vote that have annotated ¼ of the trees that are used for evaluation.

Looking on these numbers, the first astonishing fact is the stability of the results independently of the number of sentences that are used for evaluation. Put differently: With only one tree to annotate, we already get a reasonable estimate of the student's capacities.

¼ have to be annotated				
Scoring algorithm part of sentences used:	Nr of students in	Precision	Recall	**F-score**
½ (24)	31	0.9465	0.9419	**0.9439**
¼ (12)	31	0.9472	0.9379	**0.9421**
1/8 (6)	39	0.9505	0.937	**0.9433**
1/48 (1)	42	0.9512	0.9405	**0.9454**

We also checked whether the threshold (of taking only evaluations into account that are based on a reasonable number of annotated sentences) has an impact on the results, but in fact the differences are very slight. This is astonishing when looking at the annotation quality seen in section 4.4, but can be explained by the stabilizing factor that *most* students try to do a good job.

½ have to be annotated				
Adding algorithm part of sentences used:	Nr of students in	Precision	Recall	**F-score**
½ (24)	19	0.945	0.9371	**0.9408**
¼ (12)	24	0.9495	0.94	**0.9444**
1/8 (6)	28	0.948	0.9357	**0.9414**
1/48(1)	42	0.9512	0.9405	**0.9454**

1/10 have to be annotated				
Adding algorithm part of sentences used:	Nr of students in	Precision	Recall	**F-score**
½ (24)	45	0.9464	0.9409	**0.9434**
¼ (12)	40	0.9476	0.9333	**0.9399**
1/8 (6)	50	0.9476	0.9333	**0.9399**
1/48(1)	42	0.9512	0.9405	**0.9455**

Unsurprisingly, not voting but just taking the best student for each tree gives quite unstable results, depending on the number of sentences annotated by the best students. The results are partly better, partly worse than the previous results.

1/10 have to be annotated				
Meritocracy algorithm part of sentences used:	Nr of students in	Precision	Recall	**F-score**
½ (24)	1 of 45	0.9702	0.9409	**0.9749**
¼ (12)	1 of 40	0.9704	0.9634	**0.9668**
1/8 (6)	1 of 50	0.8778	0.8538	**0.8647**
1/48(1)	1 of 42	0.8407	0.8403	**0.8396**

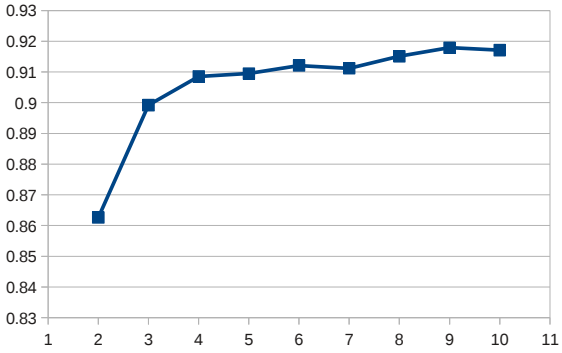
Of course it is unrealistic to have this many annotations per sentence. This leads us naturally to the exploration of how many annotations we actually need to keep up reasonable results.

4.6 Students to Quality

For a real-world annotation setting, we need to test systematically how many annotations we need for the required annotation quality.

We explored the range from 2 to 10 annotations per sentence by choosing arbitrarily for each sentence the annotators among the students that annotated the sentence (i.e. they attributed a governor to at least 80% of the words). We computed this score for 10 random attributions of each number of annotators. The results are reported in the diagram below.

On our data, the quality seems to quickly stabilize between 91 and 92% F-score. As we have seen, with higher numbers of annotators we don't get much beyond 94%. 4 or 5 annotations per sentence seems to be a reasonable number to obtain an F-score well-beyond 90%.



Average F-score
over 10 random attributions

5 Conclusion and outlook

In this paper we have presented the different features of the Arborator, a state-of-the-art online tool for collaborative dependency annotation. We have shown how most design choices were natural consequences of the annotation requirements.

We then showed that the application of the rover technique can give surprisingly good results, even though syntactic annotation is commonly considered as a task which is difficult to crowdsource (Munro et al. 2010). The reason is probably that the “crowd” is partly trained and the voting technique only has to pick out the “trained” good students. However, the data-set is too small and specific to draw more general conclusions.

We must also point out that an f-score of 0.94 and an average length of 25 tokens per sentence

means that there are on average 1.5 errors per sentence, a result which is better than most automatic annotation on out of domain data but nothing we would want to call gold-standard. But then again, this is before any bootstrapping or pedagogical improvements taking into account the typical errors – it is a very good result for a first try.

While it seems practically impossible to use the Arborator in a "real" crowdsourcing task à la Mechanical Turk because the necessary training time is excessively high, it is possible to imagine crowd-sourcing of bootstrapping techniques in dependency syntax, too. It even seems easier than for phrase structure to find non-ambiguous paraphrases that Turks could vote on in order to decide between two equally probable analyses a parser provides.

The present experiment was carried out on raw text, i.e. students had to draw all dependency links, including trivial links for example from a noun to its determiner. The natural next step is to try out this "pedagogical crowd-sourcing" in a complete bootstrapping setup: The speed of the students and thus the output could probably be dramatically increased using statistical parsers that indicate uncertainty. This uncertainty can be rendered graphically in order to attract the students' attention to the problematic dependency link. And the corrections, after having been voted on, can then again be used to train the parser on bigger data. However, it is possible that the results would be different because detecting errors in a pre-annotated corpus is a different task than not making those errors when starting from scratch.

Another possible improvement of the result could stem from the application of more general machine learning techniques, that would, for example include lexical information in the predictions – or syntactic functions if they were included in the study. In other words, such an improvement should result in a system where a student that regularly gets the dependency links of coordinative conjunctions like "*and"* wrong, would have less voting rights when deciding on the best analysis around these words.

References

Bohnet, Bernd, Andreas Langjahr, and Leo Wanner. "A development environment for an MTT-based sentence generator." In Proceedings of the first international conference on Natural language generation-Volume 14, pp. 260-263. Association for Computational Linguistics, 2000.

Bohnet, Bernd, Textgenerierung durch Transduktion linguistischer Strukturen. DISKI 298. AKA, Berlin (2006)

Boguslavsky, Igor, Svetlana Grigorieva, Nikolai Grigoriev, Leonid Kreidlin, and Nadezhda Frid. "Dependency treebank for Russian: Concept, tools, types of information." In Proceedings of the 18th conference on Computational linguistics-Volume 2, pp. 987-991. Association for Computational Linguistics, 2000.

Brants, Thorsten, and Oliver Plaehn. "Interactive corpus annotation." In LREC'00. 2000.

Chen, Xinying, Xu Chunshan, Li Wenwen. "Extracting Valency Patterns of Word Classes from Syntactic Complex Networks." Proceedings of Depling 2011, Barcelona.

De La Clergerie, Éric Villemonte. "A collaborative infrastructure for handling syntactic annotations." In proc. of The First Workshop on Automated Syntactic Annotations for Interoperable Language Resources. 2008.

Van der Beek, Leonoor, Gosse Bouma, Rob Malouf, and Gertjan Van Noord. "The Alpino dependency treebank." Language and Computers 45, no. 1 (2002): 8-22.

Fiscus, Jonathan G. "A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (ROVER)." In Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on, pp. 347-354. IEEE, 1997.

Gerdes, Kim. "Sur la non-équivalence des représentations syntaxiques: Comment la représentation en X-barre nous amène au concept du mouvement." Cahiers de grammaire 30 (2006): 175-192.

Gerdes, Kim, and Sylvain Kahane. "Defining dependencies (and constituents)." In Proceedings of the International Conference on Dependency Linguistics, Depling 2011, pp. 17-27. 2011.

Gerdes, Kim, Sylvain Kahane, Anne Lacheret, Arthur Truong, and Paola Pietrandrea. "Intonosyntactic data structures: the Rhapsodie treebank of spoken French." In Proceedings of the Sixth Linguistic Annotation Workshop, pp. 85-94. Association for Computational Linguistics, 2012.

Hajič, Jan, Barbora Vidová-Hladká, and Petr Pajas. "The prague dependency treebank: Annotation structure and support." Proceedings of the IRCS Workshop on Linguistic Databases. 2001.

Hajič, Jan. 2005. "Complex corpus annotation: The Prague dependency treebank." Insight into the Slovak and Czech Corpus Linguistics (2005): 54.

Kakkonen, Tuomo. 2006. DepAnn - An Annotation Tool for Dependency Treebanks. Proceedings of

the 11th ESSLLI Student Session at the 18th European Summer School in Logic, Language and Information (ESSLLI 2006), pp. 214-225. Malaga, Spain

Mel′čuk, Igor′ Aleksandrovič. Dependency syntax: theory and practice. State University of New York Press, 1988.

Mille, S., Burga, A., Vidal, V., & Wanner, L. (2009). Towards a rich dependency annotation of Spanish corpora. Proceedings of SEPLN'09, 325-333.

Munro, R., Bethard, S., Kuperman, V., Lai, V. T., Melnick, R., Potts, C., ... & Tily, H. (2010, June). Crowdsourcing and language studies: the new generation of linguistic data. Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk (pp. 122-130). Association for Computational Linguistics.

Mazziotta, Nicolas. "Building the syntactic reference corpus of medieval French using notabene rdf annotation tool." In Proceedings of the Fourth Linguistic Annotation Workshop, pp. 142-146. Association for Computational Linguistics, 2010.

Stenetorp, Pontus, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. "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, pp. 102-107. Association for Computational Linguistics, 2012.

Rambow, Owen, et al. "A dependency treebank for English." Proceedings of LREC. Vol. 2. 2002.

Seraji, Mojgan, and Joakim Nivre. 2012. "Bootstrapping a Persian Dependency Treebank." Linguistic Issues in Language Technology 7.1.