Arne Neumann

Merging and validating heterogenous, multi-layered corpora with `discoursegraphs`

Abstract

We present `discoursegraphs`, a library and command-line application for the conversion and merging of linguistic annotations written in Python. The software reads and writes numerous formats for syntactic and discourse-related annotations, but also supports generic interchange formats. `discoursegraphs` models primary data and its annotations as a graph and is therefore able to merge multiple independent, possibly conflicting annotation layers into a unified representation. We show how this approach is beneficial for the revision and validation of a corpus with multiple conflicting, independently annotated layers.

1 Introduction

Linguistic annotations are produced using a plethora of tools. Most of these tools focus on one type of annotation or were even developed with a specific corpus or research project in mind and use their own file formats.

To ensure that the annotations are usable beyond the lifespan of the original annotation tool or project, we might consider to convert the dataset into an interchange format that is more suitable for long-term archival, such as FoLiA (van Gompel and Reynaert, 2013), GrAF (Ide and Suderman, 2007) or PAULA (Dipper, 2005). We might also want to transform annotations to make use of modern corpus visualisation and query tools like `brat` (Stenetorp et al., 2012) or `ANNIS` (Krause and Zeldes, 2014). Both tools can visualise several independent annotation layers at the same time, but rely on custom file formats not supported by most annotation tools.[1]

While it is possible to write direct converters for all the formats we would like to map, this is not only time consuming, but also leads to information loss, e.g. when converting from a syntax representation that allows edge labels or secondary edges to one that does not.

It is therefore more desirable to use an intermediate representation that is theory neutral and capable of handling many different types of annotation.[2] This not only drastically reduces the number of converters that need to be written but also enables us

[1] For `ANNIS`, this problem is solved by `SaltNPepper` (Zipser and Romary, 2010), a conversion framework provided by the same group of developers.

[2] This is not a new idea. Graph-based intermediate representations of linguistic annotations were proposed at least as early as Ide et al. (2003).

to serialise this intermediate model into an interchange format from which the original annotation files could be recreated without losing information.

To achieve this we developed `discoursegraphs`, a converter and merging tool for linguistically annotated corpora. Via its intermediate, graph-based object representation, the software is able to transform data between a number of syntax and discourse-related annotation formats.[3] Furthermore, `discoursegraphs` allows the user to merge several independently produced and potentially conflicting layers of annotation into a unified graph representation. The tool is implemented in Python and available under the open source BSD license from its repository website.[4]

The remainder of this paper is structured as follows. Section 2 introduces the graph-based data structures used in `discoursegraphs` and their application in data conversion. In Section 3, we describe how to merge data from different annotation formats and types into a single graph. Section 4 shows how to use this merged representation to simplify the process of validating and revising heterogeneous annotations of a multi-layered corpus. Section 5 concludes the paper and suggests paths for further research.

2 Graph-based modeling of annotated corpora

Using graphs to represent linguistically annotated corpora goes back at least to Bird and Liberman (1999), who showed that a wide range of existing linguistic annotation formats and tools had a "common conceptual core" which could be represented as *annotation graphs*. This way, it becomes possible to separate the model of an annotation from its serialisation format. Such a model expresses the logical structure of an annotation and may range from the very informal (e.g. a drawing on a blackboard) to the very formal, e.g. a visualisation defined in a markup language like the ISO-standardised UML (ISO/IEC 19501, 2005).

Over the years, a number of graph-based models and accompanying formats for the representation of linguistic annotations have been proposed, i.a. PAULA (Dipper, 2005), LAF/GrAF (Ide and Suderman, 2007; ISO 24612, 2012), TCF (Heid et al., 2010), Salt (Zipser and Romary, 2010) and FoLiA (van Gompel and Reynaert, 2013).[5]

What all of these graph-based representations have in common is that they can model multiple layers of (conflicting) annotations including phenomena like overlapping and non-contiguous spans as well as multi-rooted trees – phenomena which are particularly hard or even impossible to express with inline XML, e.g. TigerXML (Mengel and Lezius, 2000), or column-based formats such as CoNLL (Hajič et al., 2009).

[3] In other words, `discoursegraphs` is neither build upon an existing linguistic exchange format nor does it offer any new intermediate format. Instead, the library reads documents in existing formats into Python objects representing property graphs and manipulates those graphs to convert the input documents into other existing formats.

[4] `https://github.com/arne-cl/discoursegraphs`. It can also be installed via Python's pip package manager and as a docker container.

[5] A comparison is beyond the scope of this paper. For a very thorough overview see Stührenberg (2012).

The question which model or format one should choose given all these similarities is not easy to answer. Good evaluation criteria might be:

Model specificity: TCF (an XML format for data exchange between linguistic web services) is very specific in that it defines a number of annotation layers that it supports (e.g. constituency parsing and discourse connectives) but it is not intended to add custom layers, e.g. for handling Rhetorical Structure Theory (Mann and Thompson, 1988) or Abstract Meaning Representation (Banarescu et al., 2013). GrAF, on the other hand, is intentionally abstract. The format allows you to encode any type of annotation that can be expressed using nodes, directed edges and arbitrary labels (in the form of feature structures that can be attached to both nodes and edges). This freedom comes with a price, as it leaves the burden to interpret the meaning of annotations to tool developers, cf. Neumann et al. (2013). Both `SaltNPepper` and `discoursegraphs` adopt a middle position, allowing users to build custom annotation layers but at the same time force them to type each edge (choosing only from a small predefined set of relation types, cf. Section 2.1). This way, tools can visualise and query previously unseen layers of annotation in a meaningful way.

Tool support: Formats like GrAF were conceived as abstract, broadly applicable standards, for which numerous converters, but no annotation, visualisation or query tools exist. In contrast, formats like FoLiA and Salt were introduced with specific corpora or projects in mind and are supported by a wider range of tools (built primarily for those corpora or projects). In addition to the type and number of tools available for a format, one might also consider the programming language the tools are implemented in when looking for the most suitable annotation model or format for one's project. With interpreted languages like Python, it is generally easier to manipulate, query and visualise the annotated data interactively[6] than it is with compiled languages like C++ and Java.[7]

We will now introduce the specific graph model used in `discoursegraphs` and give an overview over the formats it can currently read and write.

2.1 Data structures in `discoursegraphs`

The `discoursegraphs` toolkit is implemented on top of `NetworkX` (Hagberg et al., 2008), which is a graph library implemented in pure Python.[8] `discoursegraphs` is based on

[6]This is hugely facilitated by interactive computing environments like `Jupyter` (Pérez and Granger, 2007) that provide much of the ground work needed.

[7]Interactive computing in compiled languages is nevertheless possible, e.g. with `Cling` for C++ (`https://root.cern.ch/cling`) or JVM-based languages like Scala and Clojure which are interoperable with source code written in Java.

[8]Other graph libraries available for Python like `igraph` (Csardi and Nepusz, 2006) and `graph-tool` (Peixoto, 2015) are implemented in C and C++, respectively. While this makes them computationally more efficient, it also makes them harder to install, extend and debug, at least from the perspective of a software that is aimed at users with a background in linguistics rather than software engineering.

the property graph model (Rodriguez and Neubauer, 2010), which is also used in graph databases like `neo4j`[9].

A property graph is a directed, labeled and attributed multi-graph. More formally, it can be represented as a tuple $G = (V, E, R, S, \Sigma, \lambda_V, \lambda_E, \mu)$ with

V	a set of vertices (i.e. nodes)
$E \subseteq (V \times V)$	a set of directed edges
R	a set of attribute keys
S	a set of attribute values
Σ	a finite alphabet of labels
$\lambda_V : V \rightarrow \Sigma$	a mapping from a node to a node label
$\lambda_E : E \rightarrow \Sigma$	a mapping from an edge to an edge label
$\mu : (V \cup E) \times R \rightarrow S$	a mapping from nodes/edges and keys to values

While node labels (usually strings or integers) are used to uniquely identify a node within a graph, edge labels (or edge types as they are called in `discoursegraphs`) enable us to express different kinds of (linguistic) relationships between nodes (e.g. dominance vs. coreference).

Allowing for multiple edges between nodes gives us the flexibility to model multiple layers of annotations, say phrase structure and rhetorical structure, over the same primary data. For example, there might be two nodes representing constituents in the syntax layer, while the same nodes may represent the nucleus and satellite of a relation in the rhetorical structure layer.[10]

Key-value pairs (also known as attribute-value pairs) make it possible to attach arbitrary information to nodes and edges, e.g. to add part-of-speech and lemma annotations to a token node. To make the software as broadly applicable as possible, we added optional namespaces for keys. This allows us, for example, to annotate tokens with multiple part-of-speech annotations from different tag sets, e.g. `penn:pos=vbz` vs. `brown:pos=doz`.[11] Namespaces are also used to merge multiple possibly conflicting annotations over the same nodes.

One particularly important attribute we use in `discoursegraphs` is the layer. Each node and edge is assigned to at least one layer, usually named after the type of annotation (e.g. `syntax`), the format (e.g. `tiger`) or tool the data was imported from (e.g. `mmax`).

Layer attributes are particularly useful to limit the scope of a query. Like other key-value pairs, layers can use namespaces for instance to distinguish multiple layers of annotation imported from the same tool (e.g. `mmax:coreference` vs. `mmax:informationstatus`).

[9] `http://neo4j.com/`

[10] For a different example, see Figure 3 where two nodes are connected by two different edges, one dominance relation between the constituents NP and N, and another edge representing a span relation between a named entity annotation node and the token node that it covers.

[11] Here, `penn:pos=vbz` means that the token was annotated with the tag `vbz` and that this tag is used in the `penn` namespace (i.e. the Penn Treebank tagset). `brown:pos=doz` refers to the tag `doz` in the Brown Corpus tagset. Both tags represent a third person singular verb in the present tense.

The main data structure of the discoursegraphs library is the DiscourseDocumentGraph, which represents a document (e.g. a newspaper article or any other contiguous passage of written text) and all the annotations made to it.

A document can have any number of annotations (even if they are of the same type, e.g. multiple, potentially conflicting syntax trees produced by different parsers or human annotators), as long as they refer to the same tokenisation of the primary text. While this limits the applicability of the framework for corpora that depend on multiple tokenisations (e.g. diachronic and parallel corpora), it allows us to easily check heterogeneous, multi-layered annotations for consistency (see Section 4).

A DiscourseDocumentGraph contains a set of nodes and a set of directed edges. Nodes can represent tokens or any higher order structure built on top of them e.g. spans of tokens or (interior) nodes in a phrase structure tree. The directed edges signal relationships between those nodes. In discoursegraphs, we distinguish between four types of edges (cf. Figure 1):

Spanning relation: A span groups adjacent tokens into a logical unit, e.g. when annotating a phrase, named entity or other multi-word expression. A spanning relation consists of a span node with outgoing edges to each of the token nodes belonging to that span.

Dominance relation: While spans are used for 'flat' annotations applied to contiguous tokens, the concept of dominance is used to express a broad range of hierarchical relations between annotation nodes. For example, dominance relations hold i.a. between categories and subcategories in a constituency parse tree, between the head and its dependants in a dependency parse tree or between nucleus and satellite in a rhetorical structure tree.

Pointing relation: Pointers express a non-hierarchical relation between two nodes. They can e.g. be used to link an anaphor to its antecedent in a coreference relation or to signal a secondary edge, i.e. a syntactic relationship between nodes that are not in a direct dominance relation.

Precedence relation: In syntax trees and other tree-like annotations, the topological order of all nodes can be inferred from the order of the tokens in the annotated text. If the same annotations are represented as a directed graph, the ordering information gets lost unless it is explicitly encoded. This is done by adding precedence relations between each token and the token that succeeds it (as well as one precedence relation from the document root node to the first token node, thereby forming a *directed path*).

Using only these four basic edge types, a wide range of linguistic phenomena can be modeled in a single graph data structure. DiscourseDocumentGraphs allow parallel edges, i.e., there can be more than one edge from node u to node v. Having typed edges gives us a number of advantages over untyped linguistic annotation models such as LAF[12]:

Conversion without ambiguity: When converting between two annotation formats with similar use cases (e.g. two treebank formats), it might not be necessary to type the annotations, as both formats will have means to deal with both token order and hierarchy. When converting data from a generic annotation format (used for archival purposes) to a domain-specific format though, this understanding is lost. Without typed edges, a converter cannot know what kind of relation two linked nodes are in, cf. Zipser and Romary (2010); Neumann et al. (2013).

Generic visualisation: Although it is very common to draw graphs, they are, in essence, abstract mathematical structures with many possible ways to visualise them (*graph isomorphisms*). In order to visualise a large variety of linguistic annotations, we could either implement many domain-specific visualisations or simply rely on the edge types to implement just a few visualisations that are ‘good enough’ for most purposes, cf. Krause and Zeldes (2014). Precedence relations tell us the order of the leaf nodes (tokens) in our graph, which we will align horizontally. Dominance relations express hierarchy and can therefore be used to align nodes vertically. Pointing relations are non-hierarchical (i.e. they can link any two nodes of the graph) and are therefore best drawn using curved lines to avoid collisons with (straight) hierarchical edges.

Expressive queries: Without typed edges, we could only use very generic graph queries on our annotation graphs (e.g. “Does node A have a path to node B of length n?”). With types, we can instead ask questions that are linguistically motivated, i.e. “Does node A directly or indirectly dominate node B?”.

`discoursegraphs` provides simple indexing structures for often needed data, such as ordered lists of all tokens and root nodes of all sentences. This way, we avoid having to traverse the whole graph for simple operations like calculating the frequency distribution of the POS tags of all the tokens in a document. Here is an example of how to implement such a function[13]:

```
import discoursegraphs as dg
from collections import Counter
doc1 = dg.corpora.pcc.get_random_document()
freqdist = Counter()

for token_id in doc1.tokens:
    freqdist[doc1.node[token_id]['tiger:pos']] += 1
freqdist.most_common()
```

[12]LAF does allow typed nodes and edges, but does neither specify nor recommend any types. MASC (Ide et al., 2010) – the *de facto* reference corpus for the GrAF format – does not use edge types.
[13]Of course, this is already implemented in the library.

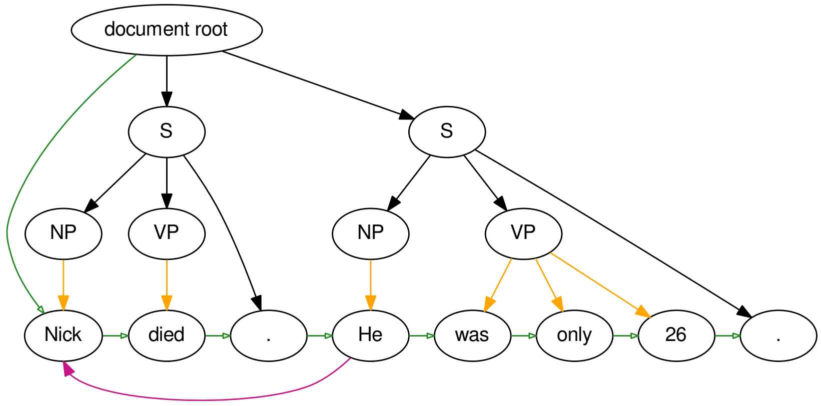


Figure 1: Example document annotated with phrase structure and coreference using all types of relations (directed edges) available in `discoursegraphs`. **Dominance relations** (black) hold between the document root node and the root nodes of the sentences it contains, as well as between constituents and subconstituents of a phrase structure. **Spanning relations** (orange) hold between preterminal nodes (constitutents) and their children (tokens). A **pointing relation** (purple) is used to signal coreference. **Precedence relations** (green) make the order of tokens in the text explicit.

Here, `doc1` is a document graph representing all the annotation layers of a single document from the PCC corpus and `doc1.tokens` is an ordered list of node IDs of all token nodes. The node IDs are then used to retrieve and count the POS tags of the tokens (i.e. values of the attribute key `tiger:pos`).

2.2 Converting between annotation formats

The data structures in `discoursegraphs` are capable of modeling many different types of annotations and are therefore well suited to act as an intermediate representation between two formats. This not only allows the library to merge multiple layers into a single graph for joint analysis, it also drastically reduces the amount of converters needed. While we would need to implement up to $n^2 - n$ converters for mapping *directly* to and from n formats, we only need $2n$ converters (one importer to and one exporter from the *intermediate representation* for each format).[14] So far, `discoursegraphs` can

[14] As one reviewer correctly pointed out, $n^2 - n$ is a purely theoretical upper limit. In practice, one would rather chain several converters than implement a direct converter for every possible combination of formats. This way, fewer converters would need to be implemented but each additional conversion step increases the risk of losing data.

import corpora from the following tools and formats:

(i.) constituent and dependency structures: Tiger-XML (Mengel and Lezius, 2000), Penn Treebank (Prasad et al., 2008) and CoNLL 2009/2010 (Hajič et al., 2009; Farkas et al., 2010)

(ii.) rhetorical structure: `RSTTool`'s (O'Donnell, 2000) rs3 and rst/dis formats

(iii.) pointing relations (e.g. coreference, connectives): formats from the `MMAX2` (Müller and Strube, 2006) and `ConAno` (Stede and Heintze, 2004) annotation tools

(iv.) annotations of spans of text: `EXMARaLDA` (Schmidt, 2004).

The library also comes with a number of exporters that can be used to convert the data into formats used by other visualisation and (linguistic) analysis tools:

(i.) general purpose graph formats like dot (Ellson et al., 2002), GEFX[15], GML[16] and GraphML (Brandes et al., 2013)

(ii.) linguistic interchange formats CoNLL 2009 and PAULA XML 1.1 (Zeldes et al., 2013),

(iii.) the geoff format of the `neo4j` graph database

(iv.) `EXMARaLDA`'s exb format.

3 Merging annotation layers

In `discoursegraphs`, it is possible to work with different annotation layers of the same text (e.g. a constituent parse tree and coreference information) individually, such that each layer or file is parsed into its own `DiscourseDocumentGraph`. This is useful for exploring a specific annotation layer, e.g. if the user needs to find out how certain attributes are named. The library tries to normalise attributes names (e.g. part-of-speech annotations are always called `pos` and never `POS`, tokens are referred to as `token` and not `tok`), but in case of doubt one can simply run the `info()` function on any `DiscourseDocumentGraph`. It will show how many nodes and edges are present in a graph, as well as the attributes they have and the (sub)layers they belong to.

If, on the other hand, the user wants to explore interactions between independent layers or files, she will need to merge them into a single graph. In Figure 2 the graph representations of two different annotation layers of the same sentence are shown. Each node has an ID, and contains a number of attributes (`begin` and `end` signal the character offsets of a token node or of the token nodes that this node dominates or spans).

[15] `http://gexf.net/format/`

[16] `http://www.fim.uni-passau.de/en/fim/faculty/chairs/theoretische-informatik/projects.html`

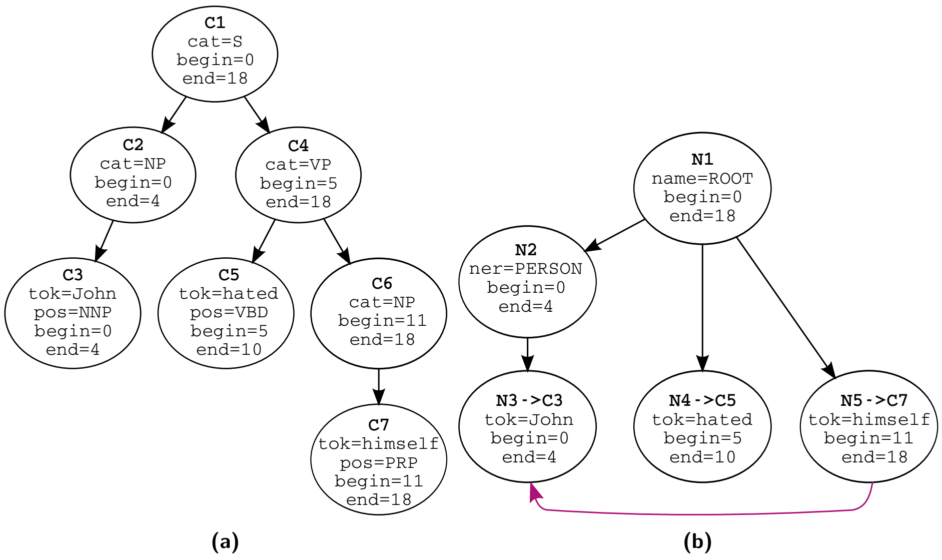


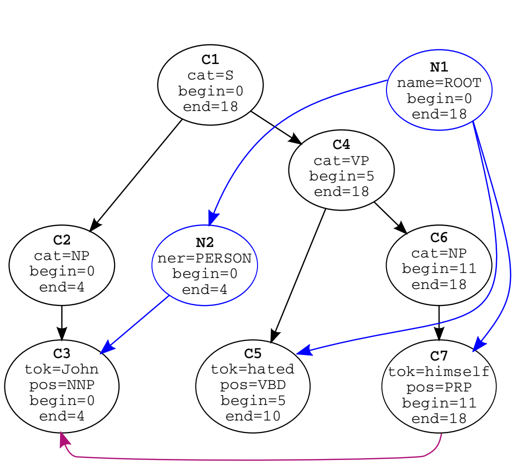
Figure 2: Two simplified document graphs representing the constituent parse tree (a), as well as the named entity and coreference annotation (b) of the sentence "*John hates himself*". The coreference is signaled by a pointing relation (purple).

Generally speaking, when merging two graphs, `discoursegraphs` will keep the first one (Figure 2a) mostly unchanged, while the nodes and edges of the second one (Figure 2b) are renamed and moved to match the first one.

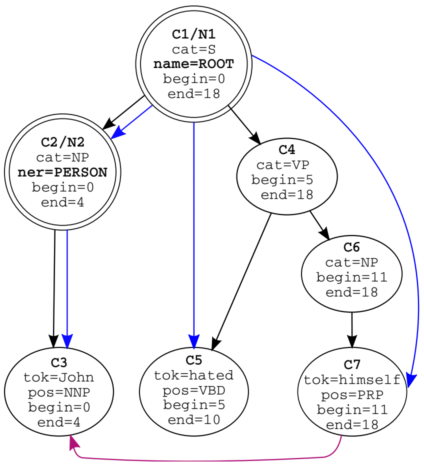
As the first step, the token nodes of the second graph will be renamed to match the tokens of the first one. Since `discoursegraphs` requires annotation layers to use the same tokenisation for merging, we can simply iterate over all token nodes in both graphs simultaneously to do so. Figure 2b shows the node IDs before and after this step, e.g. the token `himself` has the ID `N5` before renaming and `C7` afterwards.

In the second step, all nodes and edges from the second graph will be added to the first one (Figure 3a, new elements are highlighted in blue). Nodes with the same ID will be merged, i.e. additional attributes from the second graph's node will be added to the first one.

Finally, non-token nodes that cover the same span of tokens are merged. Merged nodes are drawn with a double circle, with attributes added from the second graph highlighted in bold. Note that edges are not merged, as they can carry different meanings. For example, in Figure 3b we can see two edges between the nodes `C2` and `C3`, one representing a dominance relation between a noun phrase and a proper noun `John` and the other representing a spanning relation between a `PERSON` named entity annotation and the token that it covers (also `John`).



(a) A graph containing the constituent parse tree, as well as the named entity and coreference annotation of the example sentence.



(b) A *merged* graph containing the constituent parse tree, as well as the named entity and coreference annotation of the example sentence.

Figure 3: A simplified document graph of the sentence "*John hates himself*" containing multiple layers of annotation, before and after merging annotation nodes that cover the same span. Elements added from the second graph are drawn in blue.

With this merged graph, we could now analyse interactions between syntax and coreference, between syntax and named entities or between coreference and named entities, respectively.

4 Validating heterogeneous annotation layers against each other

The graph merging facilities of `discoursegraphs` can be used for corpus maintenance, i.e. for finding errors in existing annotations and ensuring the consistency of the data across independently annotated layers.

We successfully employed the software to revise the multi-layered Potsdam Commentary Corpus (PCC, Stede (2004); Stede and Neumann (2014)). In its current version, it contains syntax, coreference, connectives and rhetorical structure annotations of 176 German newspaper commentaries. The corpus was not created as the result of a funded project and has seen many (small) contributions over the course of more than a decade. Annotators often worked on their own computers and were even allowed to edit files manually (i.e. with a text editor instead of a dedicated annotation tool). Access to version control systems could not be taken for granted and annotation guidelines were updated over the years. Inevitably, this has lead to some problems which had to be addressed during the revision.

First, we had to rename all files from all annotation layers according to the same naming scheme and ensure that they use the same encoding and line endings. Afterwards, we were able to automatically parse the different annotation files for a given newspaper commentary to check whether they used identical tokenisations.[17]

This way, we found several types of tokenisation inconsistencies:

Intentionally altered tokens: Spelling mistakes were manually corrected (at times with comments) by some annotators, but not on all layers.

Unintentionally altered tokens: Some tokens contained soft-hyphens instead of regular hyphens or featured diacritics where there should be none.

Missing or added tokens: These were mostly caused by manually corrected grammatical errors that were not present in all layers.

After fixing the tokenisation in the original source files, we leveraged the graph data structure to find errors in the annotations with respect to their connectivity:

Unreachable nodes: We found a number of tokens in the syntax annotation that were not connected to the constituency parse tree.

Superfluous edges: Coreference is usually annotated in chains, but we found numerous entities that were part of several coreference chains. In the case depicted in Figure 4, this can be either explained as an annotation error (i.e. the annotator wanted to build a chain but overlooked the nearest preceding coreferent entity) or tell us that not all entities of this "coreference cluster" are strictly coreferent. In Figure 4, this is the case for "Chancellor and Foreign Minister", who are members of "the government" but do not represent it as a whole.

Errors in manually edited, XML-based annotation files could often be found by forcing the XML parser to be strict, hereby finding elements that were never closed or closed too early.[18] Additionally, misspelled attribute names can be discovered by looking for infrequently occurring ones.

While revising a corpus becomes easier with `discoursegraphs`, it is still a laborious task that cannot be fully automated. With the knowledge that we gained and the tools that are available today, many of the issues we faced could have been avoided with a few precaution steps:

Primary data: Always keep the original data, so it never becomes necessary to guess which changes might have happened along the way.

[17]The original HTML files from which the corpus was sourced were no longer available, so our best guess was to compare against the files that contained the tokenised but otherwise unannotated commentaries.

[18]This is the default in `discoursegraphs` for all XML-based formats.

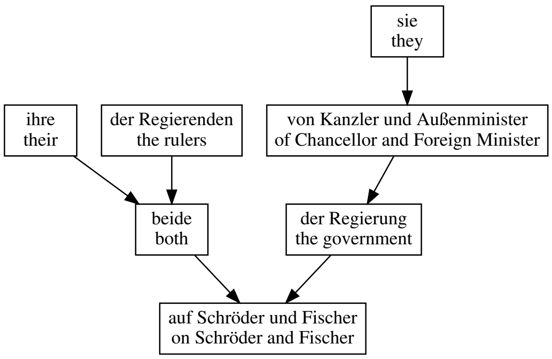


Figure 4: A graph representing three (wrongly) annotated coreference chains which share some nodes. If we accept that "Chancellor and Foreign Minister" and "the government" are coreferent, all the entities in this graph should be merged into one coreference chain.

Version control: This will help to clarify who made which changes and which annotation guidelines were used at that time.

Web-based annotation tools: With modern annotation tools like `brat`, `WebAnno` (Yimam et al., 2013) and `rstWeb` (Zeldes, 2016), annotators don't have access to the source files, so they cannot manipulate them.

Automatic validation: It is vital to provide means to validate the data automatically, e.g. via Document Type Definitions (DTDs) or XML Schema Definitions (XSDs). Such definitions can also be learned post-hoc from existing XMLs with tools like the `XML-Schema-learner`[19] (Nordmann, 2011), which allow researchers to correct existing inconsistencies.

Error guidelines: It needs to be clearly stated how annotators should deal with "errors" in the collected primary texts. In historic corpora, for example, there is a clear distinction between diplomatic and normalised transcription of the primary data.

5 Conclusion and future work

We have presented `discoursegraphs`, a library for the conversion and merging of linguistic annotations and have shown how this software facilitates the merging, revision and validation of multi-layered corpora.

In the future, we would like to add performance-oriented graph backends (e.g. igraph or graph-tool), so that a user of the library can profit from the easy debugability of `NetworkX` during the implementation of an importer or exporter, but use faster backends

[19] `https://github.com/kore/XML-Schema-learner`

later on in production. We will also try to provide a web interface for the most common functions of the library, as users without (Python) programming experience can so far only access the format conversion capabilities via the current command-line interface.

References

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *In Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*. ACL.

Bird, S. and Liberman, M. (1999). A formal framework for linguistic annotation. technical report ms-cis-99-01. Technical report, Linguistic Data Consortium, University of Pennsylvania.

Brandes, U., Eiglsperger, M., Lerner, J., and Pich, C. (2013). Graph markup language (GraphML). In Tamassia, R., editor, *Handbook of Graph Drawing and Visualization*. CRC Press.

Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9.

Dipper, S. (2005). XML-based Stand-off Representation and Exploitation of Multi-Level Linguistic Annotation. In *Berliner XML Tage*, pages 39–50.

Ellson, J., Gansner, E., Koutsofios, L., North, S. C., and Woodhull, G. (2002). Graphviz–open source graph drawing tools. In *Graph Drawing*, pages 483–484. Springer.

Farkas, R., Vincze, V., Móra, G., Csirik, J., and Szarvas, G. (2010). The CoNLL-2010 shared task: learning to detect hedges and their scope in natural language text. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning—Shared Task*, pages 1–12. Association for Computational Linguistics.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., et al. (2009). The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–18. Association for Computational Linguistics.

Heid, U., Schmid, H., Eckart, K., and Hinrichs, E. W. (2010). A corpus representation format for linguistic web services: The d-spin text corpus format and its relationship with iso standards. In *LREC*.

Ide, N., Fellbaum, C., Baker, C., and Passonneau, R. (2010). The manually annotated sub-corpus: A community resource for and by the people. In *Proceedings of the ACL 2010 conference short papers*, pages 68–73. Association for Computational Linguistics.

Ide, N., Romary, L., and de la Clergerie, E. (2003). International standard for a linguistic annotation framework. In *Proceedings of the HLT-NAACL 2003 workshop on Software engineering and architecture of language technology systems-Volume 8*, pages 25–30. Association for Computational Linguistics.

Ide, N. and Suderman, K. (2007). GrAF: A graph-based format for linguistic annotations. In *Proceedings of the Linguistic Annotation Workshop*, pages 1–8. Association for Computational Linguistics.

ISO 24612 (2012). *Language Resource Management – Linguistic Annotation Framework*. International Standards Organization, Geneva, Switzerland.

ISO/IEC 19501 (2005). *Information technology – Open Distributed Processing – Unified Modeling Language (UML)*. International Standards Organization, Geneva, Switzerland.

Krause, T. and Zeldes, A. (2014). ANNIS3: A new architecture for generic corpus query and visualization. *Literary and Linguistic Computing*.

Mann, W. C. and Thompson, S. A. (1988). Rhetorical Structure Theory: Toward a functional theory of text organization. *Text*, 8(3):243–281.

Mengel, A. and Lezius, W. (2000). An XML-based Representation Format for Syntactically Annotated Corpora. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC 2000)*.

Müller, C. and Strube, M. (2006). Multi-level annotation of linguistic data with MMAX2. In Braun, S., Kohn, K., and Mukherjee, J., editors, *Corpus technology and language pedagogy: New resources, new tools, new methods*, pages 197–214. Peter Lang.

Neumann, A., Ide, N., and Stede, M. (2013). Importing MASC into the ANNIS linguistic database: A case study of mapping GrAF. In *Proceedings of the Seventh Linguistic Annotation Workshop (LAW)*, pages 98–102. Association for Computational Linguistics.

Nordmann, K. (2011). Algorithmic learning of XML Schema definitions from XML data. Diploma thesis, Technische Universität Dortmund, Dortmund, Germany.

O'Donnell, M. (2000). RSTTool 2.4: a markup tool for Rhetorical Structure Theory. In *Proceedings of the 1st International Conference on Natural Language Generation (INLG 2000)*, pages 253–256. Association for Computational Linguistics.

Peixoto, T. P. (2015). The graph-tool python library. $https://figshare.com/articles/graph_tool/1164194$.

Pérez, F. and Granger, B. E. (2007). IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29.

Prasad, R., Dinesh, N., Lee, A., Miltsakaki, E., Robaldo, L., Joshi, A. K., and Webber, B. L. (2008). The Penn Discourse TreeBank 2.0. In *Proceedings of LREC 2008*.

Rodriguez, M. A. and Neubauer, P. (2010). Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41.

Schmidt, T. (2004). Transcribing and annotating spoken language with EXMARaLDA. In *Proceedings of the LREC-Workshop on XML based richly annotated corpora, Lisbon*, pages 69–74.

Stede, M. (2004). The Potsdam Commentary Corpus. In *Proceedings of the 2004 ACL Workshop on Discourse Annotation*, pages 96–102. Association for Computational Linguistics.

Stede, M. and Heintze, S. (2004). Machine-assisted Rhetorical Structure Annotation. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, pages 425–431. Association for Computational Linguistics.

Stede, M. and Neumann, A. (2014). Potsdam Commentary Corpus 2.0: Annotation for Discourse Research. In *Ninth International Conference on Language Resources and Evaluation*, Reykjavik, Iceland. European Language Resources Association (ELRA).

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 Session at EACL 2012*, Avignon, France. Association for Computational Linguistics.

Stührenberg, M. (2012). *Auszeichnungssprachen für linguistische Korpora: theroretische Grundlagen, De-facto-Standards, Normen.* PhD thesis, Bielefeld University.

van Gompel, M. and Reynaert, M. (2013). FoLiA: A practical XML Format for Linguistic Annotation-a descriptive and comparative study. *Computational Linguistics in the Netherlands Journal*, 3:63–81.

Yimam, S. M., Gurevych, I., de Castilho, R. E., and Biemann, C. (2013). WebAnno: A Flexible, Web-based and Visually Supported System for Distributed Annotations. In *ACL (Conference System Demonstrations)*, pages 1–6.

Zeldes, A. (2016). rstWeb–A Browser-based Annotation Interface for Rhetorical Structure Theory and Discourse Relations. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 1–5.

Zeldes, A., Zipser, F., and Neumann, A. (2013). PAULA XML Documentation: Format Version 1.1. Research Report, hal-00783716, https://hal.inria.fr/hal-00783716.

Zipser, F. and Romary, L. (2010). A model oriented approach to the mapping of annotation formats using standards. In *Workshop on Language Resource and Language Technology Standards, LREC 2010.*